The Complete
Python & C++ Manual

Expert Tutorials to Improve Your Skills

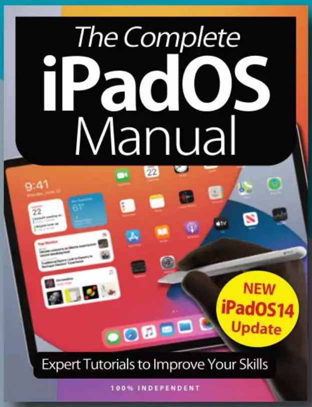
100% INDEPENDENT

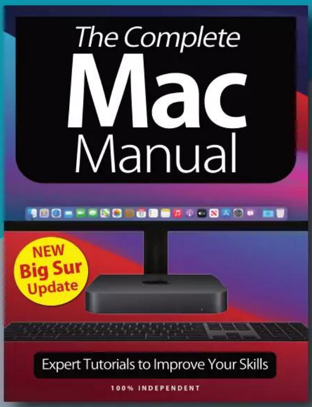
VISIT…

LANZAROTE
Caliente.COM

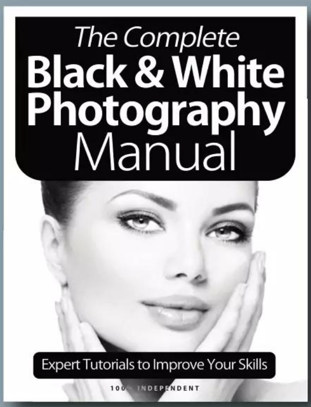
Discover more of our guides today...

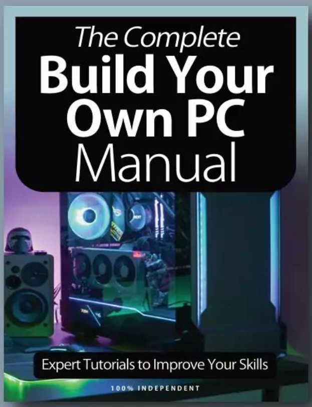
The Complete
iPhone
Manual
NEW iOS 14 Update
Expert Tutorials to Improve Your Skills
100% INDEPENDENT

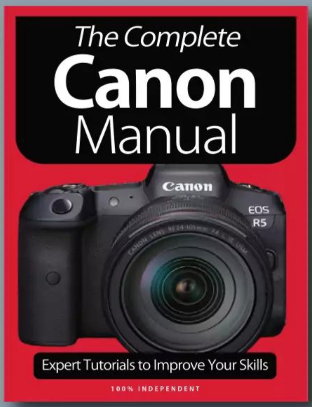

The Complete
iPadOS
Manual
NEW iPadOS 14 Update
Expert Tutorials to Improve Your Skills
100% INDEPENDENT

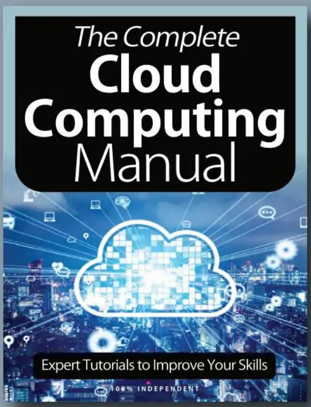

The Complete
Mac
Manual
NEW Big Sur Update
Expert Tutorials to Improve Your Skills
100% INDEPENDENT

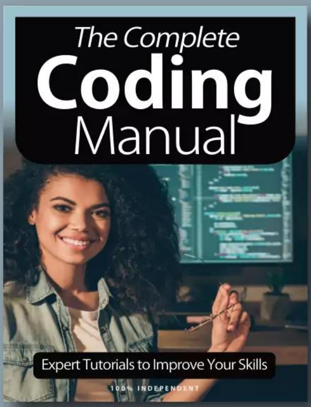
The Complete
Android
Manual
Expert Tutorials to Improve Your Skills
100% INDEPENDENT


The Complete
Black & White Photography
Manual
Expert Tutorials to Improve Your Skills
100% INDEPENDENT

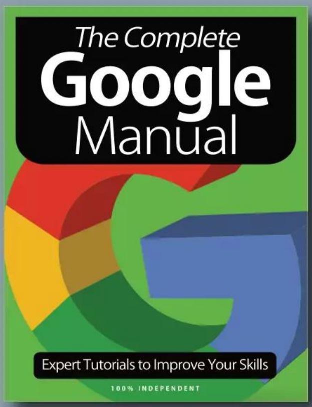

The Complete
Build Your Own PC
Manual
Expert Tutorials to Improve Your Skills
100% INDEPENDENT

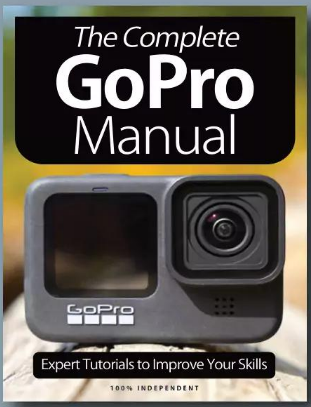

The Complete
Canon
Manual
Canon
EOS R5
Expert Tutorials to Improve Your Skills
100% INDEPENDENT


The Complete
Cloud Computing
Manual
Expert Tutorials to Improve Your Skills
100% INDEPENDENT


The Complete
Coding
Manual
Expert Tutorials to Improve Your Skills
100% INDEPENDENT

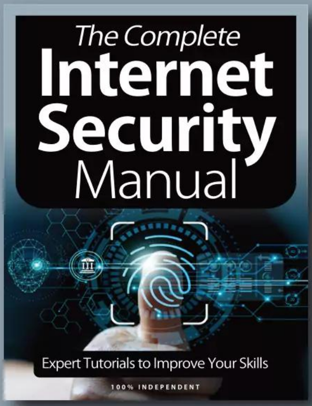
The Complete
FRITZ!Box
Manual
FRITZ!
Expert Tutorials to Improve Your Skills
100% INDEPENDENT

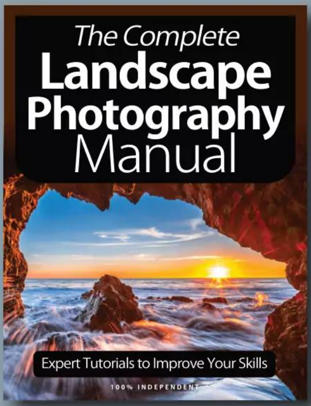

The Complete
Google
Manual
Expert Tutorials to Improve Your Skills
100% INDEPENDENT


The Complete
GoPro
Manual
GoPro
Expert Tutorials to Improve Your Skills
100% INDEPENDENT

The Complete
Instagram
Manual
Expert Tutorials to Improve Your Skills
100% INDEPENDENT

The Complete
WhatsApp
Manual
Expert Tutorials to Improve Your Skills
100% INDEPENDENT


The Complete
Internet Security
Manual
Expert Tutorials to Improve Your Skills
100% INDEPENDENT


The Complete
Landscape Photography
Manual
Expert Tutorials to Improve Your Skills
100% INDEPENDENT

The Complete Python & C++ Manual

Unleash the power of Python and C++

Having a basic knowledge of programming can open many different doors for the newcomer to explore. You can gain a better understanding of how hardware and software work together, how your computer or device functions and how incredible open-world gaming environments are converted from ones and zeros to what's on your monitor or TV.

Technology is everywhere and it's all connected through programming. Your TV, microwave, in-car entertainment and the Internet itself are all reliant on good programming to make them work the way you want them to.

Within these pages are the building blocks to help you take your first steps into the world of programming. We've taken two of the most powerful and versatile programming languages available, Python and C++ and broken them down into bite-sized tutorials and guides to help you learn how they work, and how to make them work for you.

Learn how to install them, print simple messages to the screen, ask for user input and manipulate the data to produce amazing results. By the end of this book you will understand how Python and C++ work and what potential lies beyond.

Ready? Let's get programming!

@bdmpubs BDM Publications www.bdmpublications.com

Contents

6 Say Hello to Python

8 Why Python?
10 Equipment You Will Need
12 Getting to Know Python
14 How to Set Up Python in Windows
16 How to Set Up Python on a Mac
18 How to Set Up Python in Linux

20 Getting Started with Python

22 Starting Python for the First Time
24 Your First Code
26 Saving and Executing Your Code
28 Executing Code from the Command Line
30 Numbers and Expressions
32 Using Comments
34 Working with Variables
36 User Input
38 Creating Functions
40 Conditions and Loops
42 Python Modules

44 Working with Data

46 Lists
48 Tuples
50 Dictionaries
52 Splitting and Joining Strings
54 Formatting Strings
56 Date and Time
58 Opening Files
60 Writing to Files
62 Exceptions
64 Python Graphics

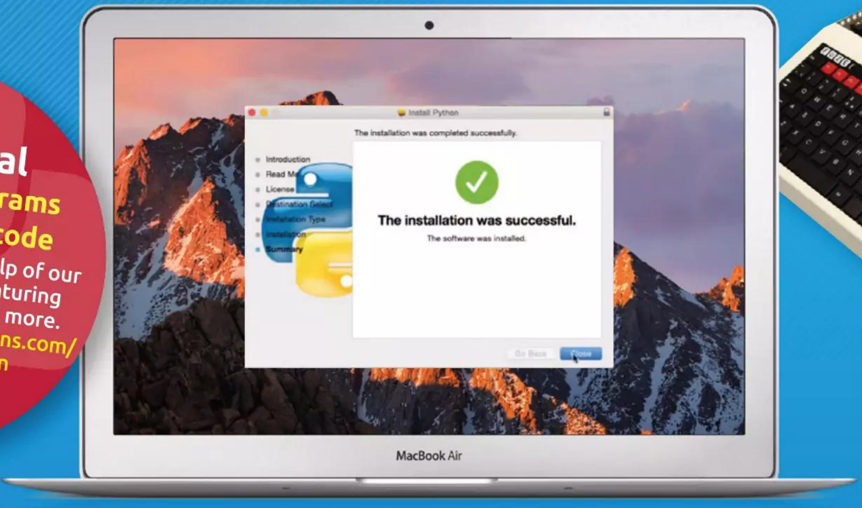
66 Using Modules

68 Calendar Module
70 OS Module
72 Random Module
74 Tkinter Module
76 Pygame Module
80 Create Your Own Modules

86 Say Hello to C++

84 Why C++?
86 Equipment Needed
88 How to Set Up C++ in Windows
90 How to Set Up C++ on a Mac
92 How to Set Up C++ in Linux
94 Other C++ IDEs to Install

BDM's Code Portal
60+ Python programs
21,500+ lines of code
Master Python with the help of our fantastic Code Portal, featuring code for games, tools and more.
Visit: https://bdmpublications.com/code-portal, and log in to get access!



96 C++ Fundamentals

98 Your First C++ Program
100 Structure of a C++ Program
102 Compile and Execute
104 Using Comments
106 Variables
108 Data Types
110 Strings
112 C++ Maths

114 C++ Input/Output

116 User Interaction
118 Character Literals
120 Defining Constants
122 File Input/Output

124 Loops and Decision Making

126 While Loop
128 For Loop
130 Do… While Loop
132 If Statement
134 If… Else Statement

136 Working with Code

138 Common Coding Mistakes
140 Beginner Python Mistakes
142 Beginner C++ Mistakes
144 Where Next?

BDM's i-Tech Special – 5th Edition – ISSN 2047-2269
Published by: ....Black Dog Media Limited (BDM)
Editor: ....James Gale
Art Director & Production: . . .Mark Ayshford
Production Manager: ....Karl Linstead
Design: ....Robin Drew, Lena Whitaker
Editorial:.....................David Hayward
Sub Editor:..................Paul Beard

Printed and bound in Great Britain by: Acorn Web Offset Ltd
Newsstand distribution by: Seymour Distribution Limited
2, East Poultry Avenue, London EC1A 9PT
International distribution by: Pineapple Media Limited
www.pineapple-media.com
Digital distribution by: Readly AB, Pocketmags, Zinio, Cafeyn, Magzter
For all advertising and promotional opportunities contact:
enquiries@bdmpublications.com

Copyright © 2021 Black Dog Media. All rights reserved.

INTERNATIONAL LICENSING – Black Dog Media has many great publications and all are available for licensing worldwide. For more information go to: www.brucesawfordlicensing.com; email: bruce@brucesawfordlicensing.com; telephone: 0044 7831 567372

Editorial and design are the copyright © Papercut Limited and is reproduced under licence to Black Dog Media. No part of this publication may be reproduced in any form, stored in a retrieval system or integrated into any other publication, database or commercial programs without the express written permission of the publisher. Under no circumstances should this publication and its contents be resold, loaned out or used in any form by way of trade without the publisher's written permission. While we pride ourselves on the quality of the information we provide, Black Dog Media Limited reserves the right not to be held responsible for any mistakes or inaccuracies found within the text of this publication. Due to the nature of the software industry, the publisher cannot guarantee that all tutorials will work on every version of Raspbian OS. It remains the purchaser's sole responsibility to determine the suitability of this book and its content for whatever purpose. Images reproduced on the front and back cover are solely for design purposes and are not representative of content. We advise all potential buyers to check listing prior to purchase for confirmation of actual content. All editorial opinion herein is that of the reviewer as an individual and is not representative of the publisher or any of its affiliates. Therefore the publisher holds no responsibility in regard to editorial opinion and content.

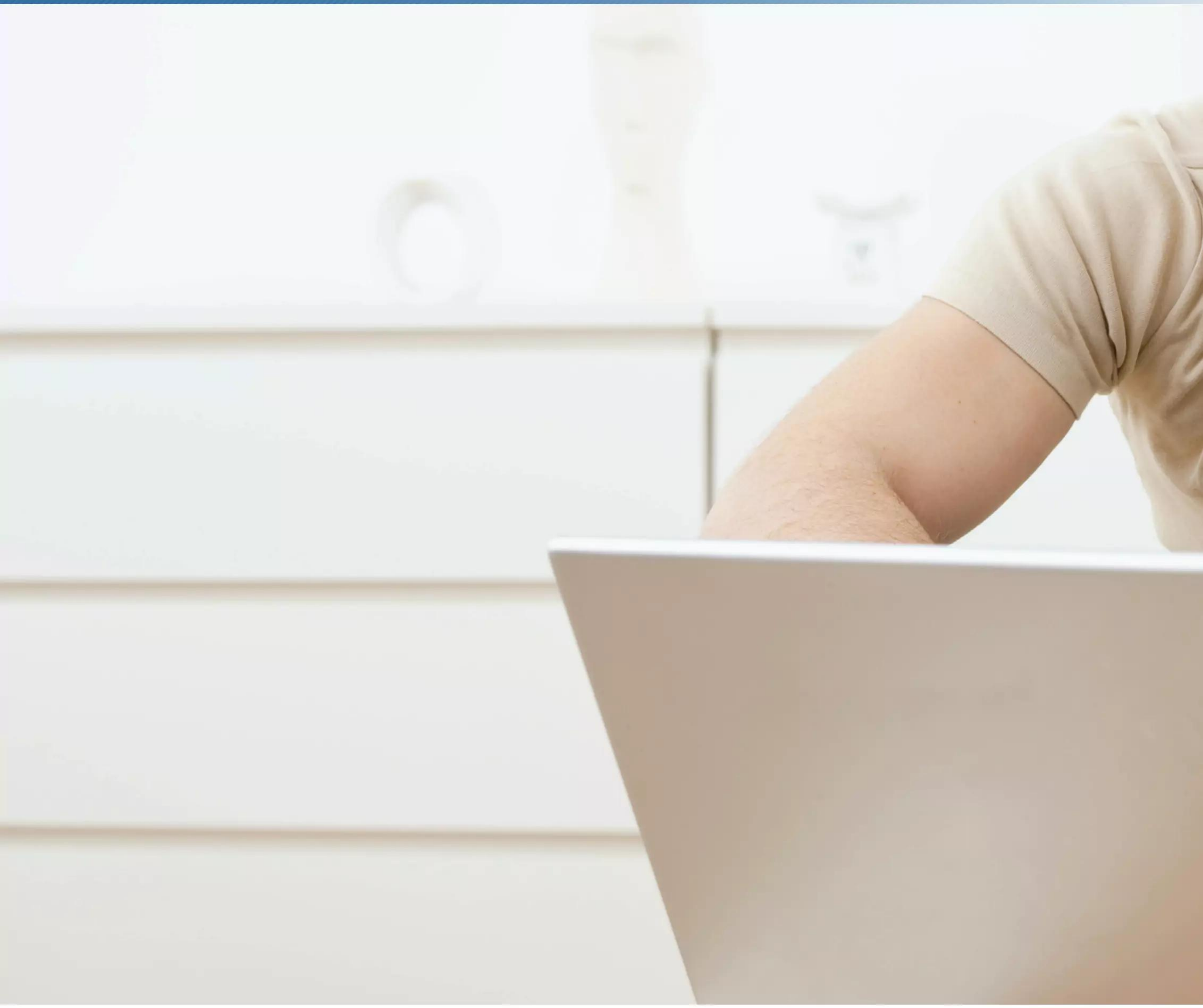
BDM's i-Tech Special is an independent publication and as such does not necessarily reflect the views or opinions of the producers contained within. This publication is not endorsed or associated in any way with The Linux Foundation, Linux Mint, The Raspberry Pi Foundation, ARM Holding, Python, Debian Project, Lenovo, Dell, Hewlett-Packard, Apple or any associate or affiliate company. All copyrights, trademarks and registered trademarks for the respective companies are acknowledged. Relevant graphic imagery reproduced with courtesy of Lenovo, Raspberry Pi, Hewlett- Packard, Dell, Python, Microsoft, Samsung and Apple.

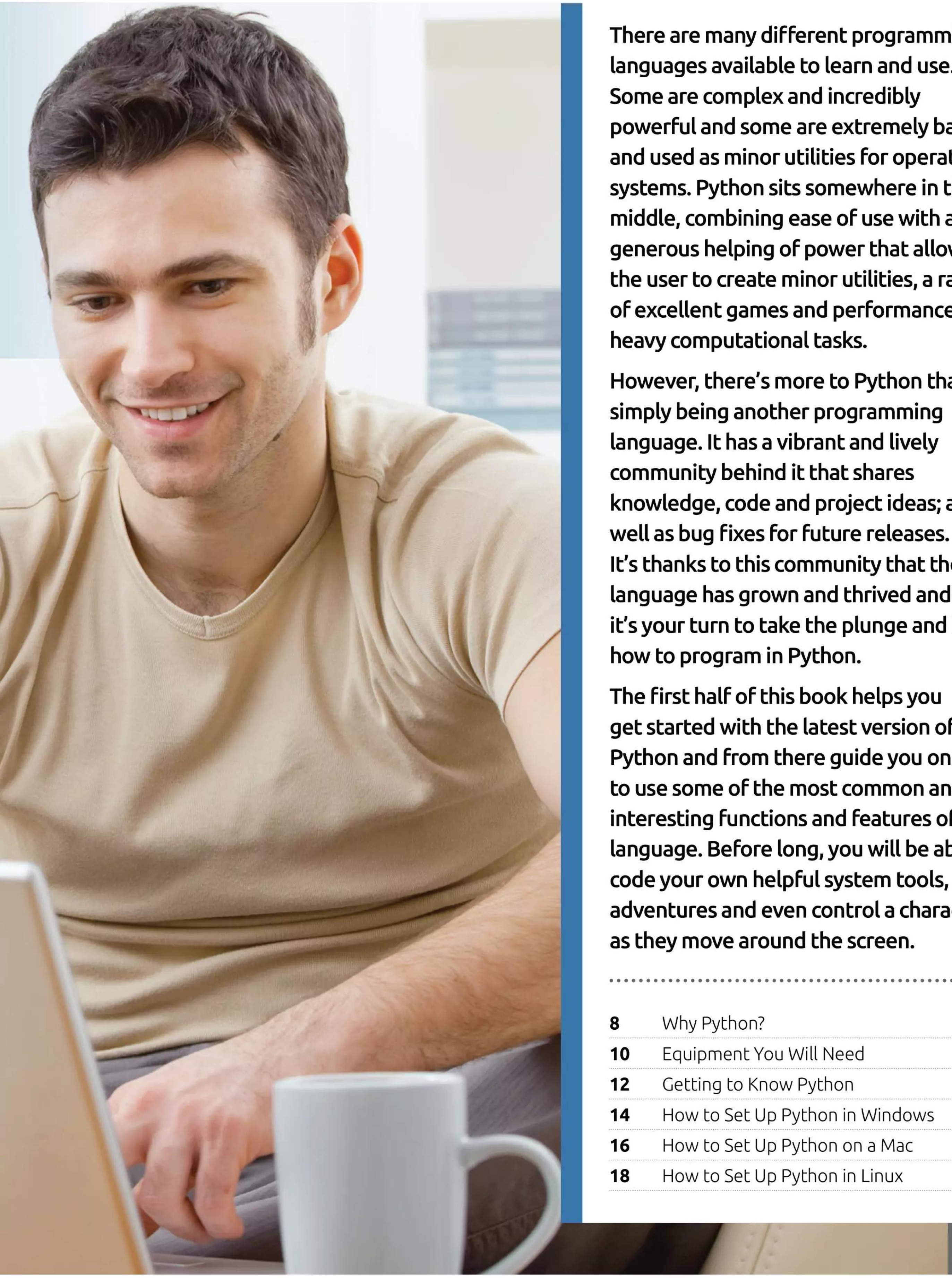
Additional images contained within this publication are reproduced under licence from Shutterstock.com.

Prices, international availability, ratings, titles and content are subject to change. All information was correct at time of print. Some content may have been previously published in other volumes or BDM titles. We advise potential buyers to check the suitability of contents prior to purchase.

Black Dog Media Limited (BDM)
Registered in England & Wales No: 5311511

Say Hello to Python





There are many different programming languages available to learn and use. Some are complex and incredibly powerful and some are extremely basic and used as minor utilities for operating systems. Python sits somewhere in the middle, combining ease of use with a generous helping of power that allows the user to create minor utilities, a range of excellent games and performance-heavy computational tasks.

However, there's more to Python than simply being another programming language. It has a vibrant and lively community behind it that shares knowledge, code and project ideas; as well as bug fixes for future releases. It's thanks to this community that the language has grown and thrived and now it's your turn to take the plunge and learn how to program in Python.

The first half of this book helps you get started with the latest version of Python and from there guide you on how to use some of the most common and interesting functions and features of the language. Before long, you will be able to code your own helpful system tools, text adventures and even control a character as they move around the screen.

8 Why Python?
10 Equipment You Will Need
12 Getting to Know Python
14 How to Set Up Python in Windows
16 How to Set Up Python on a Mac
18 How to Set Up Python in Linux

Why Python?

There are many different programming languages available for the modern computer, and some still available for older 8 and 16-bit computers too. Some of these languages are designed for scientific work, others for mobile platforms and such. So why choose Python out of all the rest?

PYTHON POWER

Ever since the earliest home computers were available, enthusiasts, users and professionals have toiled away until the wee hours, slaving over an overheating heap of circuitry to create something akin to magic.

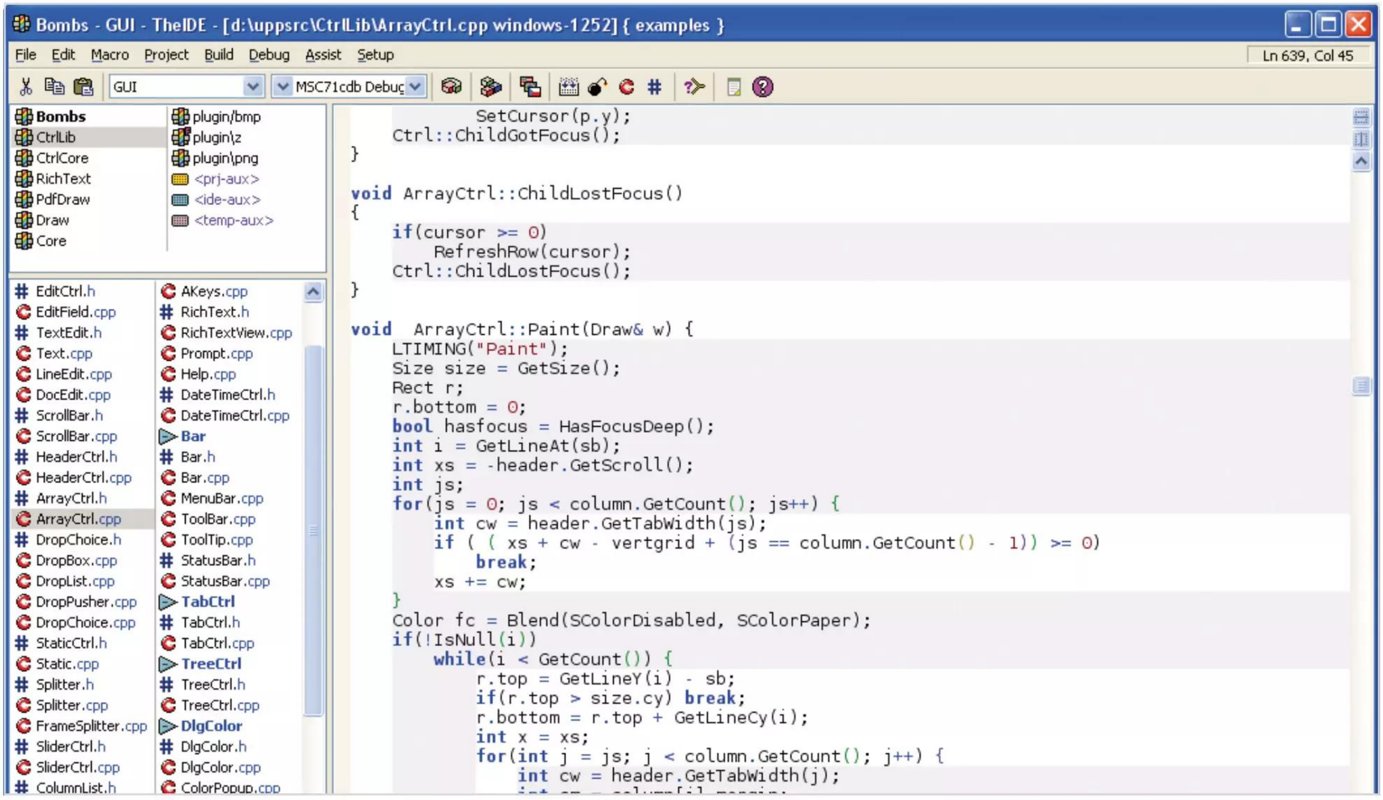
These pioneers of programming carved their way into a new frontier, forging small routines that enabled the letter 'A' to scroll across the screen. It may not sound terribly exciting to a generation that's used to ultra-high definition graphics and open world, multi-player online gaming. However, forty-something years ago it was blindingly brilliant.

Naturally these bedroom coders helped form the foundations for every piece of digital technology we use today. Some went on to become chief developers for top software companies, whereas others pushed the available hardware to its limits and founded the billion pound gaming empire that continually amazes us.

Regardless of whether you use an Android device, iOS device, PC, Mac, Linux, Smart TV, games console, MP3 player, GPS device built-in to a car, set-top box or a thousand other connected and 'smart' appliances, behind them all is programming.

All those aforementioned digital devices need instructions to tell them what to do, and allow them to be interacted with. These instructions form the programming core of the device and that core can be built using a variety of programming languages.

The languages in use today differ depending on the situation, the platform, the device's use and how the device will interact with its


Bombs - GUI - TheIDE - [d:\uppsrc\CtrlLib\ArrayCtrl.cpp windows-1252] { examples }
File
Edit
Macro
Project
Build
Debug
Assist
Setup
Ln 639, Col 45
GUI
MSC71cdb Debug
Bombs
CtrlLib
CtrlCore
RichText
PdfDraw
Draw
Core
plugin/bmp
plugin\z
plugin\png
<prj-aux>
<ide-aux>
<temp-aux>
EditCtrl.h
EditField.cpp
TextEdit.h
Text.cpp
LineEdit.cpp
DocEdit.cpp
ScrollBar.h
ScrollBar.cpp
HeaderCtrl.h
HeaderCtrl.cpp
ArrayCtrl.h
ArrayCtrl.cpp
DropChoice.h
DropBox.cpp
DropList.cpp
DropPusher.cpp
DropChoice.cpp
StaticCtrl.h
Static.cpp
Splitter.h
Splitter.cpp
FrameSplitter.cpp
SliderCtrl.h
SliderCtrl.cpp
ColumnList.h
AKeys.cpp
RichText.h
RichTextView.cpp
Prompt.cpp
Help.cpp
DateTimeCtrl.h
DateTimeCtrl.cpp
Bar
Bar.h
Bar.cpp
MenuBar.cpp
ToolBar.cpp
ToolTip.cpp
StatusBar.h
StatusBar.cpp
TabCtrl
TabCtrl.h
TabCtrl.cpp
TreeCtrl
TreeCtrl.h
TreeCtrl.cpp
DlgColor
DlgColor.h
DlgColor.cpp
ColorPopup.cpp

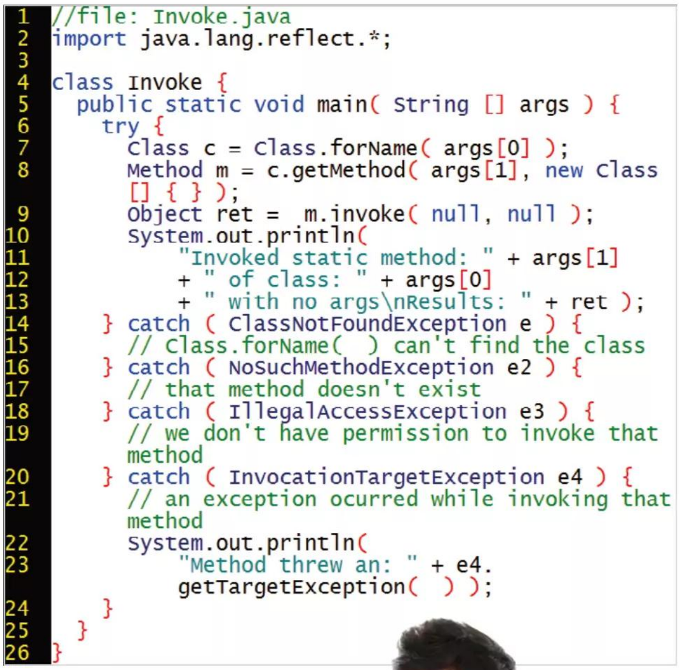
```
            SetCursor(p.y);
        Ctrl::ChildGotFocus();
}

void ArrayCtrl::ChildLostFocus()
{
        if(cursor >= 0)
            RefreshRow(cursor);
        Ctrl::ChildLostFocus();
}

void  ArrayCtrl::Paint(Draw& w) {
        LTIMING("Paint");
        Size size = GetSize();
        Rect r;
        r.bottom = 0;
        bool hasfocus = HasFocusDeep();
        int i = GetLineAt(sb);
        int xs = -header.GetScroll();
        int js;
        for(js = 0; js < column.GetCount(); js++) {
            int cw = header.GetTabWidth(js);
            if ( ( xs + cw - vertgrid + (js == column.GetCount() - 1)) >= 0)
                break;
            xs += cw;
        }
        Color fc = Blend(SColorDisabled, SColorPaper);
        if(!IsNull(i))
            while(i < GetCount()) {
                r.top = GetLineY(i) - sb;
                if(r.top > size.cy) break;
                r.bottom = r.top + GetLineCy(i);
                int x = xs;
                for(int j = js; j < column.GetCount(); j++) {
                    int cw = header.GetTabWidth(j);
```

C++ is usually reserved for more complex programs, operating systems, games and so on.

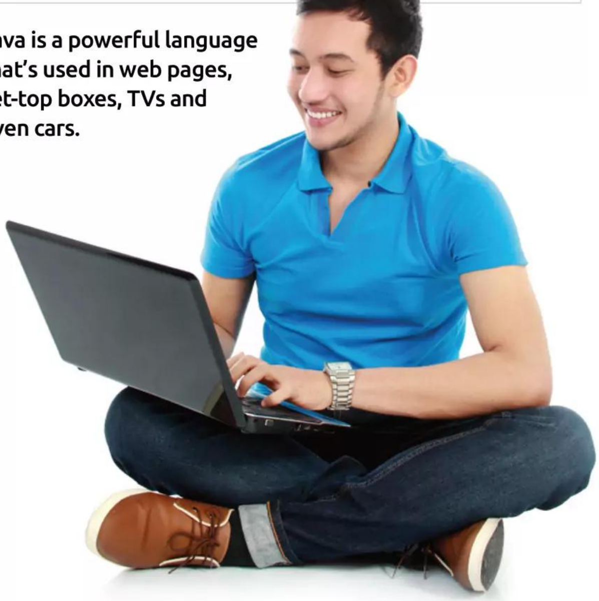
environment or users. Operating systems, such as Windows, macOS and such are usually a combination of C++, C#, assembly and some form of visual-based language. Games generally use C++ whilst web pages can use a plethora of available languages such as HTML, Java, Python and so on.

More general-purpose programming is used to create programs, apps, software or whatever else you want to call them. They're widely used across all hardware platforms and suit virtually every conceivable application. Some operate faster than others and some are easier to learn and use than others. Python is one such general-purpose language.

Python is what's known as a High-Level Language, in that it 'talks' to the hardware and operating system using a variety of arrays, variables, objects, arithmetic, subroutines, loops and countless more interactions. Whilst it's not as streamlined as a Low-Level Language, which can deal directly with memory addresses, call stacks and registers, its benefit is that it's universally accessible and easy to learn.



```
//file: Invoke.java
import java.lang.reflect.*;

class Invoke {
  public static void main( String [] args ) {
    try {
      Class c = Class.forName( args[0] );
      Method m = c.getMethod( args[1], new Class
      [] { } );
      Object ret =  m.invoke( null, null );
      System.out.println(
          "Invoked static method: " + args[1]
          + " of class: " + args[0]
          + " with no args\nResults: " + ret );
    } catch ( ClassNotFoundException e ) {
      // Class.forName( ) can't find the class
    } catch ( NoSuchMethodException e2 ) {
      // that method doesn't exist
    } catch ( IllegalAccessException e3 ) {
      // we don't have permission to invoke that
      method
    } catch ( InvocationTargetException e4 ) {
      // an exception ocurred while invoking that
      method
      System.out.println(
          "Method threw an: " + e4.
          getTargetException( ) );
    }
  }
}
```

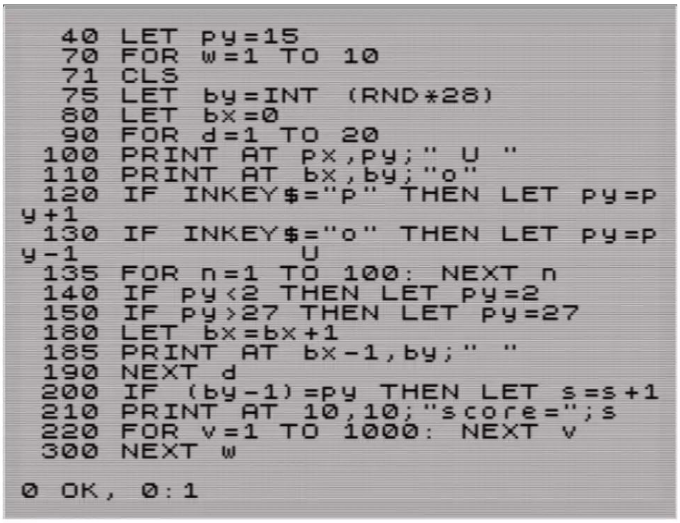


Java is a powerful language that's used in web pages, set-top boxes, TVs and even cars.

Python was created over twenty six years ago and has evolved to become an ideal beginner's language for learning how to program a computer. It's perfect for the hobbyist, enthusiast, student, teacher and those who simply need to create their own unique interaction between either themselves or a piece of external hardware and the computer itself.

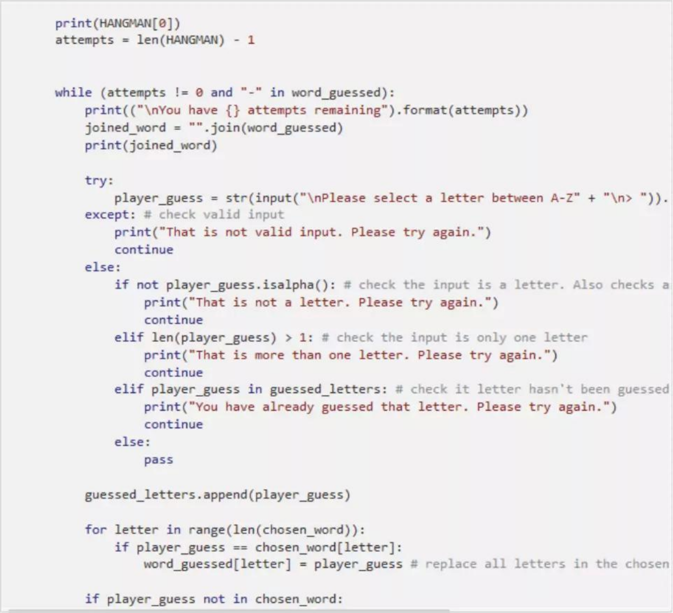
Python is free to download, install and use and is available for Linux, Windows, macOS, MS-DOS, OS/2, BeOS, IBM i-series machines, and even RISC OS. It has been voted one of the top five programming languages in the world and is continually evolving ahead of the hardware and Internet development curve.

So to answer the question: why python? Simply put, it's free, easy to learn, exceptionally powerful, universally accepted, effective and a superb learning and educational tool.



```
  40 LET py=15
  70 FOR w=1 TO 10
  71 CLS
  75 LET by=INT (RND*28)
  80 LET bx=0
  90 FOR d=1 TO 20
 100 PRINT AT px,py;" U "
 110 PRINT AT bx,by;"o"
 120 IF INKEY$="p" THEN LET py=p
y+1
 130 IF INKEY$="o" THEN LET py=p
y-1           U
 135 FOR n=1 TO 100: NEXT n
 140 IF py<2 THEN LET py=2
 150 IF py>27 THEN LET py=27
 180 LET bx=bx+1
 185 PRINT AT bx-1,by;"  "
 190 NEXT d
 200 IF (by-1)=py THEN LET s=s+1
 210 PRINT AT 10,10;"score=";s
 220 FOR v=1 TO 1000: NEXT v
 300 NEXT w

0 OK, 0:1
```

BASIC was once the starter language that early 8-bit home computer users learned.



```
print(HANGMAN[0])
attempts = len(HANGMAN) - 1


while (attempts != 0 and "-" in word_guessed):
    print(("\nYou have {} attempts remaining").format(attempts))
    joined_word = "".join(word_guessed)
    print(joined_word)

    try:
        player_guess = str(input("\nPlease select a letter between A-Z" + "\n> ")).
    except: # check valid input
        print("That is not valid input. Please try again.")
        continue
    else:
        if not player_guess.isalpha(): # check the input is a letter. Also checks a
            print("That is not a letter. Please try again.")
            continue
        elif len(player_guess) > 1: # check the input is only one letter
            print("That is more than one letter. Please try again.")
            continue
        elif player_guess in guessed_letters: # check it letter hasn't been guessed
            print("You have already guessed that letter. Please try again.")
            continue
        else:
            pass

    guessed_letters.append(player_guess)

    for letter in range(len(chosen_word)):
        if player_guess == chosen_word[letter]:
            word_guessed[letter] = player_guess # replace all letters in the chosen

    if player_guess not in chosen_word:
```

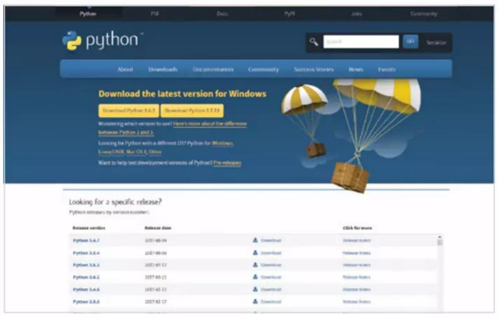
Python is a more modern take on BASIC, it's easy to learn and makes for an ideal beginner's programming language.

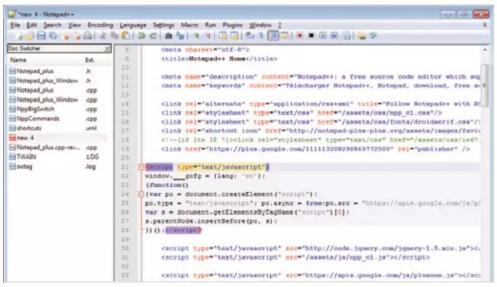
Equipment You Will Need

You can learn Python with very little hardware or initial financial investment. You don't need an incredibly powerful computer and any software that's required is freely available.

WHAT WE'RE USING

Thankfully, Python is a multi-platform programming language available for Windows, macOS, Linux, Raspberry Pi and more. If you have one of those systems, then you can easily start using Python.





☐ COMPUTER

Obviously you're going to need a computer in order to learn how to program in Python and to test your code. You can use Windows (from XP onward) on either a 32 or 64-bit processor, an Apple Mac or Linux installed PC.

☐ AN IDE

An IDE (Integrated Developer Environment) is used to enter and execute Python code. It enables you to inspect your program code and the values within the code, as well as offering advanced features. There are many different IDEs available, so find the one that works for you and gives the best results.

☐ PYTHON SOFTWARE

macOS and Linux already come with Python preinstalled as part of the operating system, as does the Raspberry Pi. However, you need to ensure that you're running the latest version of Python. Windows users need to download and install Python, which we'll cover shortly.

☐ TEXT EDITOR

Whilst a text editor is an ideal environment to enter code into, it's not an absolute necessity. You can enter and execute code directly from the IDLE but a text editor, such as Sublime Text or Notepad++, offers more advanced features and colour coding when entering code.

☐ INTERNET ACCESS

Python is an ever evolving environment and as such new versions often introduce new concepts or change existing commands and code structure to make it a more efficient language. Having access to the Internet will keep you up-to-date, help you out when you get stuck and give access to Python's immense number of modules.

☐ TIME AND PATIENCE

Despite what other books may lead you to believe, you won't become a programmer in 24-hours. Learning to code in Python takes time, and patience. You may become stuck at times and other times the code will flow like water. Understand you're learning something entirely new, and you will get there.

THE RASPBERRY PI

Why use a Raspberry Pi? The Raspberry Pi is a tiny computer that's very cheap to purchase but offers the user a fantastic learning platform. Its main operating system, Raspbian, comes preinstalled with the latest Python along with many Modules and extras.

RASPBERRY PI

The Raspberry Pi 3 is the latest version, incorporating a more powerful CPU, more memory, Wi-Fi and Bluetooth support. You can pick up a Pi for around £32 or as a part of kit for £50+, depending on the kit you're interested in.

FUZE PROJECT

The FUZE is a learning environment built on the latest model of the Raspberry Pi. You can purchase the workstations that come with an electronics kit and even a robot arm for you to build and program. You can find more information on the FUZE at **www.fuze.co.uk**.

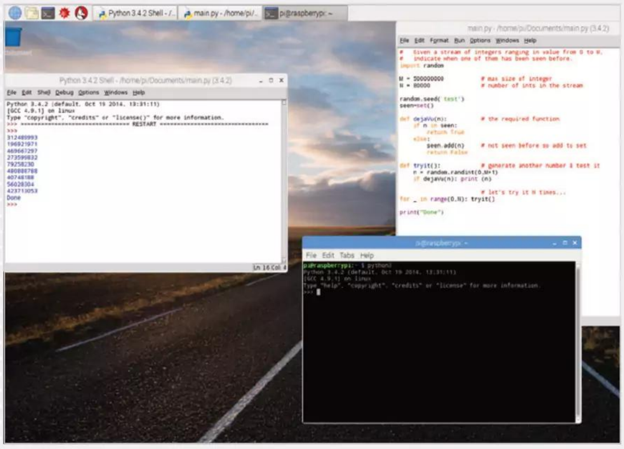
BOOKS

We have several great coding titles available via **www.bdmpublications.com**. Our Pi books cover how to buy your first Raspberry Pi, set it up and use it; there are some great step-by-step project examples and guides to get the most from the Raspberry Pi too.

Coding for Beginners
Helping You with Your First Steps in Coding
Coding Tips, Tricks & Fixes
The ultimate guide to advancing your coding skills and knowledge
Over 867 Expert Hints Inside
The python Manual
Learn how to master Python and expand your programming skills
Over 873 Python guides & tips
Raspberry Pi Tricks, Hacks & Fixes
Learn to get the most from the amazing Raspberry Pi
Fully Updated Edition
Over 192 PAGES

RASPBIAN

The Raspberry Pi's main operating system is a Debian-based Linux distribution that comes with everything you need in a simple to use package. It's streamlined for the Pi and is an ideal platform for hardware and software projects, Python programming and even as a desktop computer.


Python 3.4.2 Shell - /home/pi/Documents/main.py (3.4.2)
File Edit Shell Debug Options Windows Help
main.py - /home/pi/Documents/main.py (3.4.2)
File Edit Format Run Options Windows Help
pi@raspberrypi: ~
File Edit Tabs Help
Ln 16 Col 4

```
#   Given a stream of integers ranging in value from 0 to M,
#   indicate when one of them has been seen before.
import random

M = 500000000           # max size of integer
N = 80000               # number of ints in the stream

random.seed('test')
seen=set()

def dejaVu(n):          # the required function
    if n in seen:
        return True
    else:
        seen.add(n)     # not seen before so add to set
        return False

def tryit():            # generate another number & test it
    n = random.randint(0,M+1)
    if dejavu(n): print (n)

                        # let's try it N times...
for _ in range(0,N): tryit()

print("Done")
```

```
pi@raspberrypi:~ $ python3
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Getting to Know Python

Python is the greatest computer programming language ever created. It enables you to fully harness the power of a computer, in a language that's clean and easy to understand.

WHAT IS PROGRAMMING?

It helps to understand what a programming language is before you try to learn one, and Python is no different. Let's take a look at how Python came about and how it relates to other languages.

PYTHON

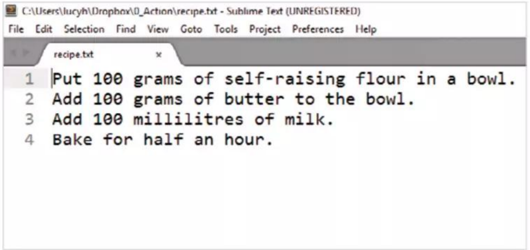
A programming language is a list of instructions that a computer follows. These instructions can be as simple as displaying your name or playing a music file, or as complex as building a whole virtual world. Python is a programming language conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language.

Guido van Rossum, the father of Python.

PROGRAMMING RECIPES

Programs are like recipes for computers. A recipe to bake a cake could go like this:

Put 100 grams of self-raising flour in a bowl.
Add 100 grams of butter to the bowl.
Add 100 millilitres of milk.
Bake for half an hour.

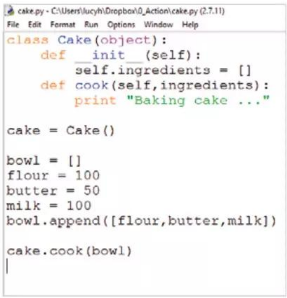

C:\Users\lucyh\Dropbox\0_Action\recipe.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
recipe.txt

```
Put 100 grams of self-raising flour in a bowl.
Add 100 grams of butter to the bowl.
Add 100 millilitres of milk.
Bake for half an hour.
```

CODE

Just like a recipe, a program consists of instructions that you follow in order. A program that describes a cake might run like this:

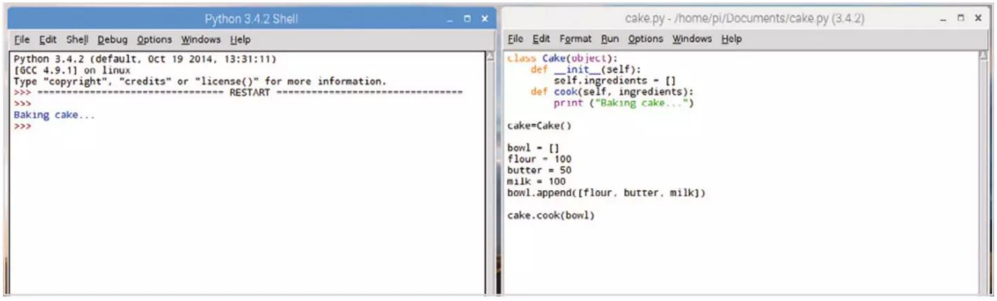
```
bowl = []
flour = 100
butter = 50
milk = 100
bowl.append([flour,butter,milk])
cake.cook(bowl)
```


cake.py - C:\Users\lucyh\Dropbox\0_Action\cake.py (2.7.11)
File Edit Format Run Options Window Help

```
class Cake(object):
    def __init__(self):
        self.ingredients = []
    def cook(self,ingredients):
        print "Baking cake ..."

cake = Cake()

bowl = []
flour = 100
butter = 50
milk = 100
bowl.append([flour,butter,milk])

cake.cook(bowl)
```

PROGRAM COMMANDS

You might not understand some of the Python commands, like bowl.append and cake.cook(bowl). The first is a list, the second an object; we'll look at both in this book. The main thing to know is that it's easy to read commands in Python. Once you learn what the commands do, it's easy to figure out how a program works.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
cake.py - /home/pi/Documents/cake.py (3.4.2)
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Baking cake...
>>>
```

```
class Cake(object):
    def __init__(self):
        self.ingredients - []
    def cook(self, ingredients):
        print ("Baking cake...")

cake=Cake()

bowl - []
flour - 100
butter = 50
milk = 100
bowl.append([flour, butter, milk])

cake.cook(bowl)
```

HIGH-LEVEL LANGUAGES

Computer languages that are easy to read are known as "high-level". This is because they fly high above the hardware (also referred to as "the metal"). Languages that "fly close to the metal," like Assembly, are known as "low-level". Low-level languages commands read a bit like this: `msg db ,0xa len equ $ - msg`.

High-level programming language

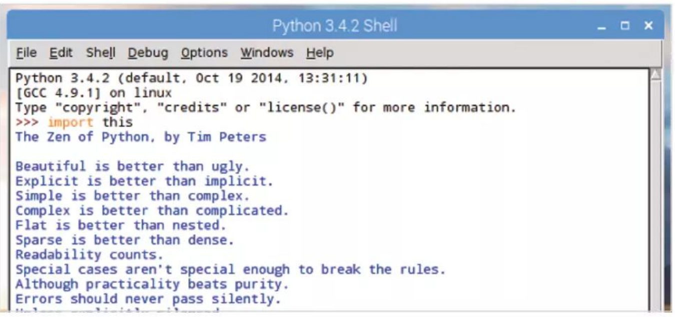
From Wikipedia, the free encyclopedia

Contents

Features [edit]

ZEN OF PYTHON

Python lets you access all the power of a computer in a language that humans can understand. Behind all this is an ethos called "The Zen of Python." This is a collection of 20 software principles that influences the design of the language. Principles include "Beautiful is better than ugly" and "Simple is better than complex." Type `import this` into Python and it will display all the principles.



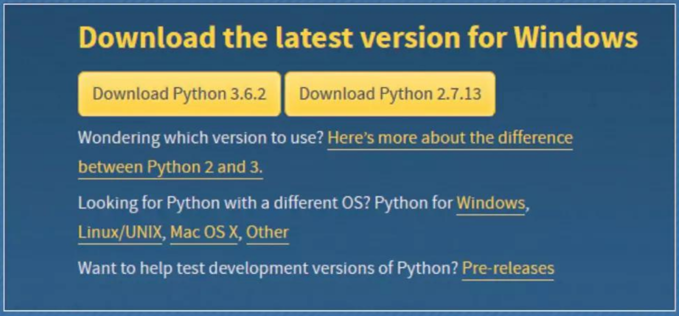
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
```

PYTHON 3 VS PYTHON 2

In a typical computing scenario, Python is complicated somewhat by the existence of two active versions of the language: Python 2 and Python 3.

WORLD OF PYTHON

When you visit the Python Download page you'll notice that there are two buttons available: one for Python 3.6.2 and the other for Python 2.7.13; correct at the time of writing (remember Python is frequently updated so you may see different version numbers).



Download the latest version for Windows

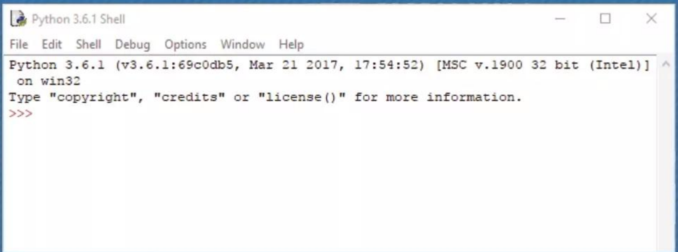
Wondering which version to use? Here's more about the difference between Python 2 and 3.

Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Pre-releases

PYTHON 3.X

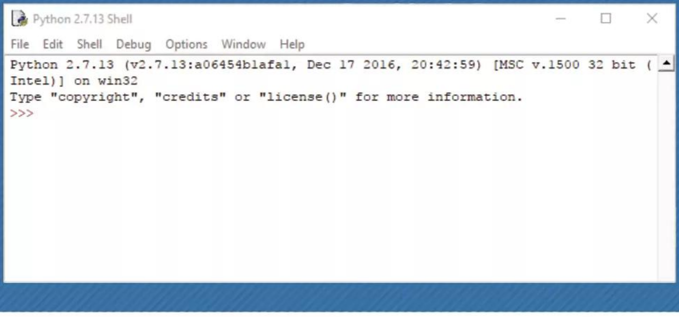
In 2008 Python 3 arrived with several new and enhanced features. These features provide a more stable, effective and efficient programming environment but sadly, most (if not all) of these new features are not compatible with Python 2 scripts, modules and tutorials. Whilst not popular at first, Python 3 has since become the cutting edge of Python programming.



```
Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 17:54:52) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

PYTHON 2.X

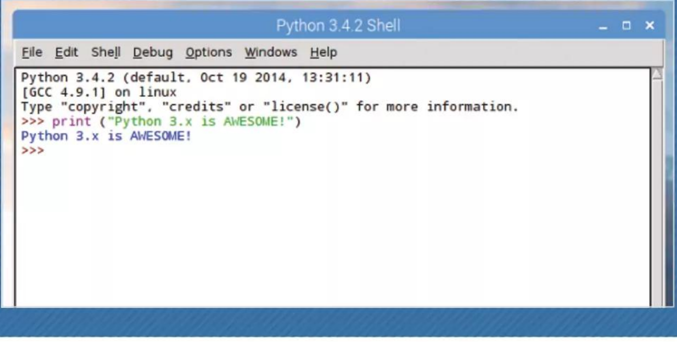
So why two? Well, Python 2 was originally launched in 2000 and has since then adopted quite a large collection of modules, scripts, users, tutorials and so on. Over the years Python 2 has fast become one of the first go to programming languages for beginners and experts to code in, which makes it an extremely valuable resource.



```
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 20:42:59) [MSC v.1500 32 bit (
Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

3.X WINS

Python 3's growing popularity has meant that it's now prudent to start learning to develop with the new features and begin to phase out the previous version. Many development companies, such as SpaceX and NASA use Python 3 for snippets of important code.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> print ("Python 3.x is AWESOME!")
Python 3.x is AWESOME!
>>>
```

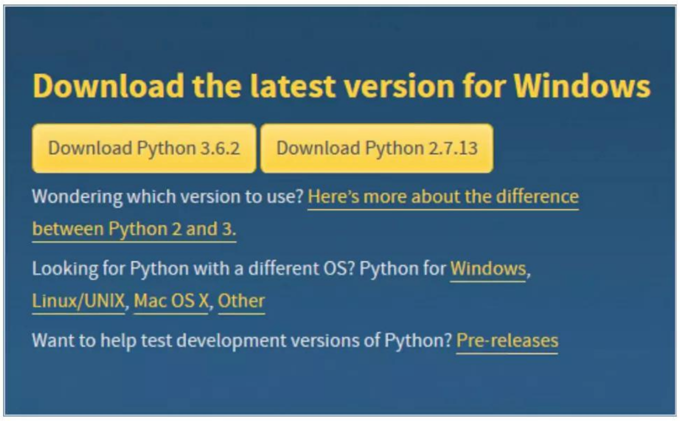
How to Set Up Python in Windows

Windows users can easily install the latest version of Python via the main Python Downloads page. Whilst most seasoned Python developers may shun Windows as the platform of choice for building their code, it's still an ideal starting point for beginners.

INSTALLING PYTHON 3.X

Microsoft Windows doesn't come with Python preinstalled as standard, so you're going to have to install it yourself manually. Thankfully, it's an easy process to follow.

STEP 1 Start by opening your web browser to **www.python.org/downloads/**. Look for the button detailing the download link for Python 3.x.x (in our case this is Python 3.6.2 but as mentioned you may see later versions of 3).



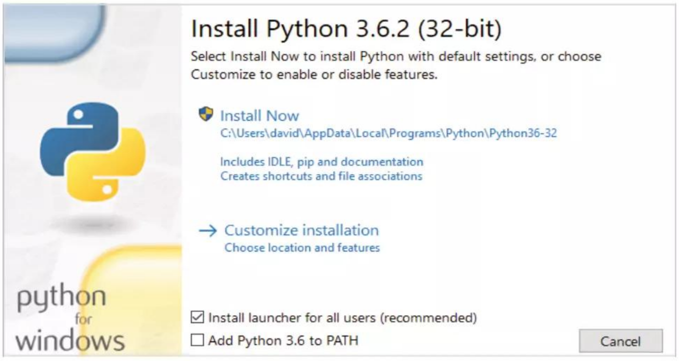
Download the latest version for Windows

Wondering which version to use? Here's more about the difference between Python 2 and 3.

Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Pre-releases

STEP 2 Click the download button for version 3.x, and save the file to your Downloads folder. When the file is downloaded, double-click the executable and the Python installation wizard will launch. From here you have two choices: Install Now and Customise Installation. We recommend opting for the Customise Installation link.



Install Python 3.6.2 (32-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

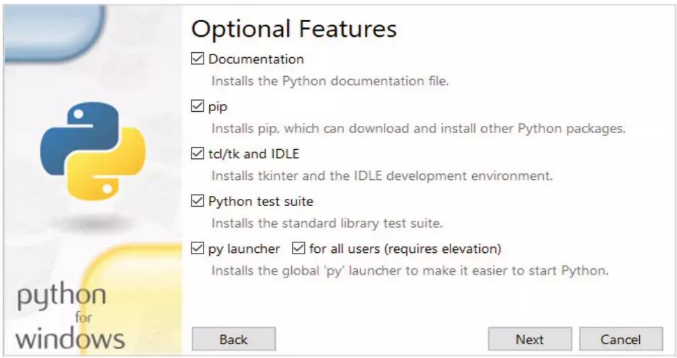
Install Now
C:\Users\david\AppData\Local\Programs\Python\Python36-32

Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☐ Add Python 3.6 to PATH

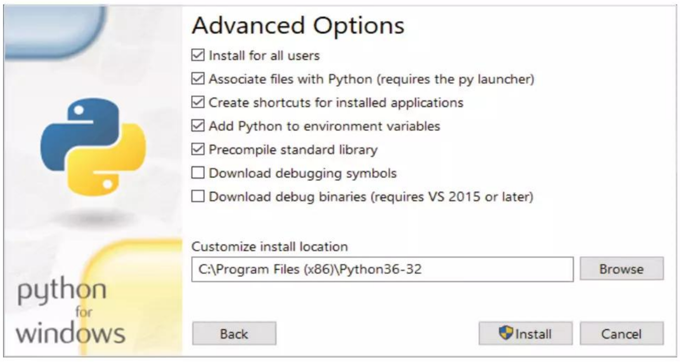
STEP 3 Choosing the Customise option allows you to specify certain parameters, and whilst you may stay with the defaults, it's a good habit to adopt as sometimes (not with Python, thankfully) installers can include unwanted additional features. On the first screen available, ensure all boxes are ticked and click the Next button.



Optional Features

☑ Documentation
Installs the Python documentation file.
☑ pip
Installs pip, which can download and install other Python packages.
☑ tcl/tk and IDLE
Installs tkinter and the IDLE development environment.
☑ Python test suite
Installs the standard library test suite.
☑ py launcher ☑ for all users (requires elevation)
Installs the global 'py' launcher to make it easier to start Python.

STEP 4 The next page of options include some interesting additions to Python. Ensure the Associate file with Python, Create Shortcuts, Add Python to Environment Variables, Precompile Standard Library and Install for All Users options are ticked. These make using Python later much easier. Click Install when you're ready to continue.



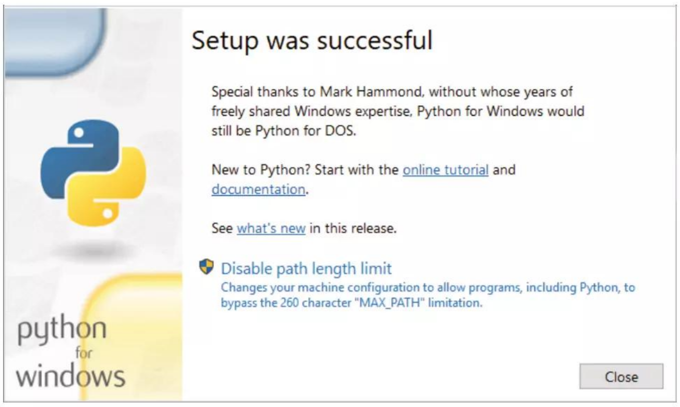
Advanced Options

☑ Install for all users
☑ Associate files with Python (requires the py launcher)
☑ Create shortcuts for installed applications
☑ Add Python to environment variables
☑ Precompile standard library
☐ Download debugging symbols
☐ Download debug binaries (requires VS 2015 or later)

Customize install location
C:\Program Files (x86)\Python36-32

STEP 5

You may need to confirm the installation with the Windows authentication notification. Simply click Yes and Python will begin to install. Once the installation is complete the final Python wizard page will allow you to view the latest release notes, and follow some online tutorials.



Setup was successful

Special thanks to Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.

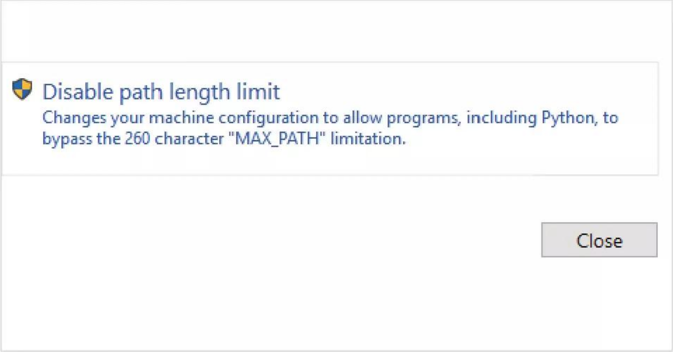
New to Python? Start with the online tutorial and documentation.

See what's new in this release.

Disable path length limit
Changes your machine configuration to allow programs, including Python, to bypass the 260 character "MAX_PATH" limitation.

STEP 6

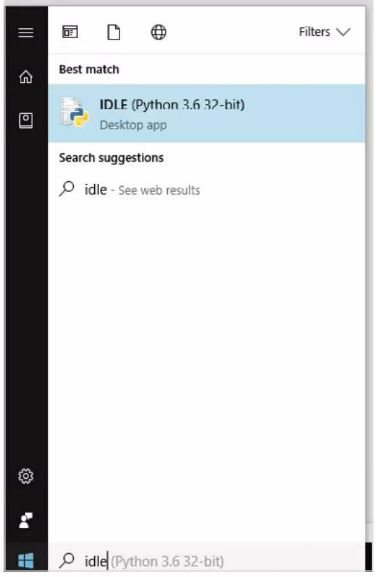
Before you close the install wizard window, however, it's best to click on the link next to the shield detailed Disable Path Length Limit. This will allow Python to bypass the Windows 260 character limitation, enabling you to execute Python programs stored in deep folders arrangements. Again, click Yes to authenticate the process; then you can Close the installation window.



Disable path length limit
Changes your machine configuration to allow programs, including Python, to bypass the 260 character "MAX_PATH" limitation.

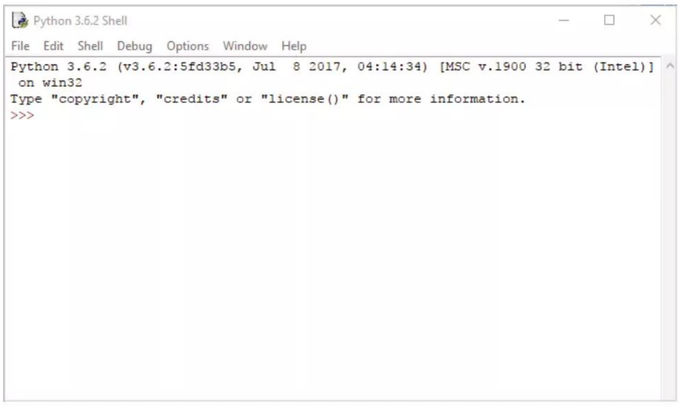
STEP 7

Windows 10 users will now find the installed Python 3.x within the Start button Recently Added section. The first link, Python 3.6 (32-bit) will launch the command line version of Python when clicked (more on that in a moment). To open the IDLE, type IDLE into Windows start.



STEP 8

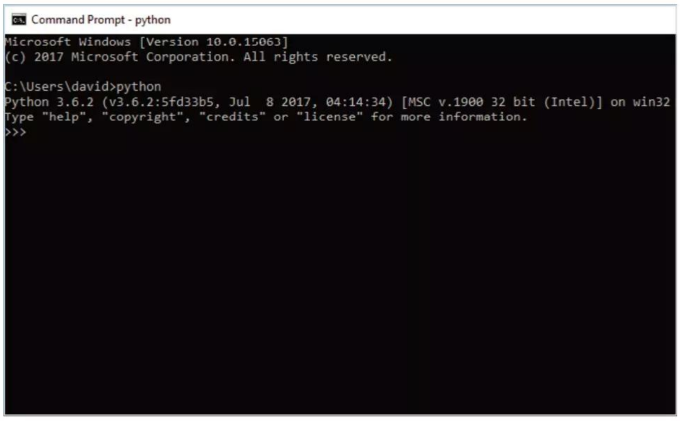
Clicking on the IDLE (Python 3.6 32-bit) link will launch the Python Shell, where you can begin your Python programming journey. Don't worry if your version is newer, as long as it's Python 3.x our code will work inside your Python 3 interface.



```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

STEP 9

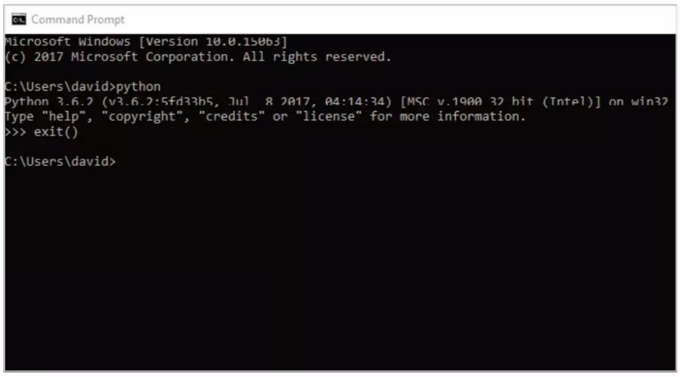
If you now click on the Windows Start button again, and this time type: `CMD`, you'll be presented with the Command Prompt link. Click it to get to the Windows command line environment. To enter Python within the command line, you need to type: `python` and press Enter.



```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\david>python
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

STEP 10

The command line version of Python works in much the same way as the Shell you opened in Step 8; note the three left-facing arrows (>>>). Whilst it's a perfectly fine environment, it's not too user-friendly, so leave the command line for now. Enter: `exit()` to leave and close the Command Prompt window.



```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\david>python
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()

C:\Users\david>
```

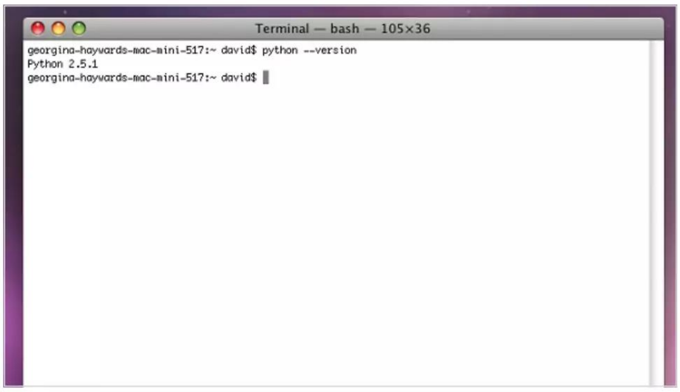
How to Set Up Python on a Mac

If you're running an Apple Mac, then setting up Python is incredibly easy. In fact a version of Python is already installed. However, you should make sure you're running the latest version.

INSTALLING PYTHON

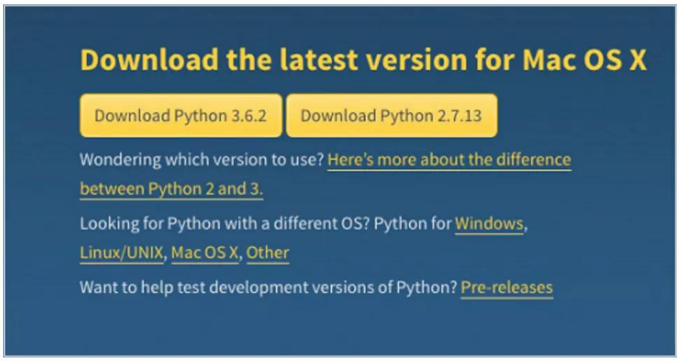
Apple's operating system comes with Python installed, so you don't need to install it separately. However, Apple doesn't update Python very often and you're probably running an older version. So it makes sense to check and update first.

STEP 1 Open a new Terminal window by clicking Go > Utilities, then double-click the Terminal icon. Now enter: `python --version`. You should see "Python 2.5.1" and even later, if Apple has updated the OS and Python installation. Either way, it's best to check for the latest version.



```
georgina-haywards-mac-mini-517:~ david$ python --version
Python 2.5.1
georgina-haywards-mac-mini-517:~ david$ 
```

STEP 2 Open Safari and head over to **www.python.org/downloads**. Just as with the Windows setup procedure on the previous pages, you can see two yellow download buttons: one for Python 3.6.2, and the other for Python 2.7.13. Note, that version numbers may be different due to the frequent releases of Python.



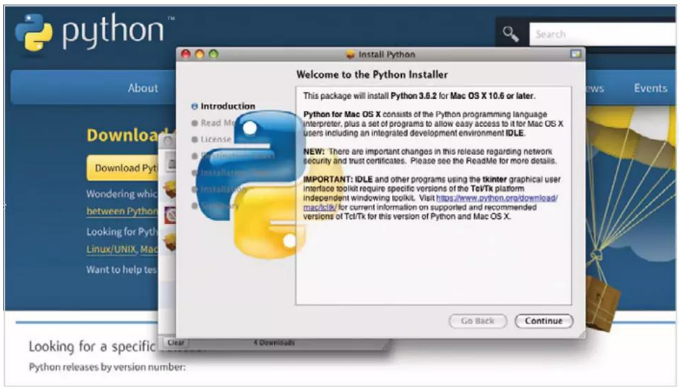
Download the latest version for Mac OS X

Wondering which version to use? Here's more about the difference between Python 2 and 3.

Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Pre-releases

STEP 3 Click on the latest version of Python 3.x, in our case this is the download button for Python 3.6.2. This will automatically download the latest version of Python and depending on how you've got your Mac configured, it automatically starts the installation wizard.



Welcome to the Python Installer

This package will install **Python 3.6.2** for **Mac OS X 10.6 or later**.

Python for Mac OS X consists of the Python programming language interpreter, plus a set of programs to allow easy access to it for Mac OS X users including an integrated development environment **IDLE**.

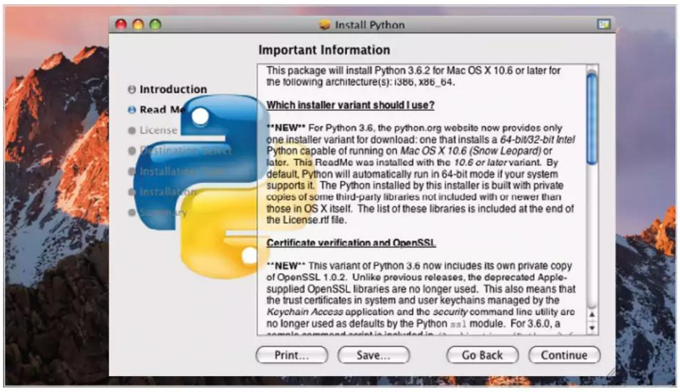
NEW: There are important changes in this release regarding network security and trust certificates. Please see the ReadMe for more details.

IMPORTANT: IDLE and other programs using the **tkinter** graphical user interface toolkit require specific versions of the **Tcl/Tk** platform independent windowing toolkit. Visit https://www.python.org/download/mac/tcltk/ for current information on supported and recommended versions of Tcl/Tk for this version of Python and Mac OS X.

Looking for a specific release?

Python releases by version number:

STEP 4 With the Python installation wizard open, click on the Continue button to begin the installation. It's worth taking a moment to read through the Important Information section, in case it references something that applies to your version of macOS. When ready, click Continue again.



Important Information

This package will install Python 3.6.2 for Mac OS X 10.6 or later for the following architecture(s): i386, x86_64.

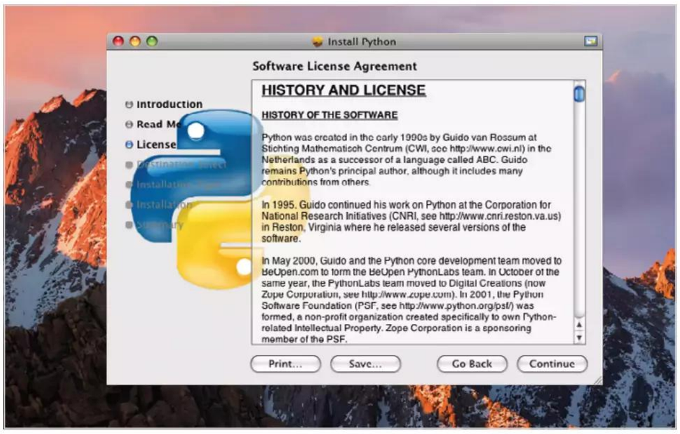
Which installer variant should I use?

NEW For Python 3.6, the python.org website now provides only one installer variant for download: one that installs a *64-bit/32-bit Intel* Python capable of running on *Mac OS X 10.6 (Snow Leopard)* or later. This ReadMe was installed with the *10.6 or later* variant. By default, Python will automatically run in 64-bit mode if your system supports it. The Python installed by this installer is built with private copies of some third-party libraries not included with or newer than those in OS X itself. The list of these libraries is included at the end of the License.rtf file.

Certificate verification and OpenSSL

NEW This variant of Python 3.6 now includes its own private copy of OpenSSL 1.0.2. Unlike previous releases, the deprecated Apple-supplied OpenSSL libraries are no longer used. This also means that the trust certificates in system and user keychains managed by the *Keychain Access* application and the *security* command line utility are no longer used as defaults by the Python `ssl` module. For 3.6.0, a

STEP 5 The next section details the Software License Agreement, and whilst not particularly interesting to most folks, it's probably worth a read. When you're ready, click on the Continue button once again.


Install Python
Introduction
Read Me
License
Print...
Save...
Go Back
Continue

Software License Agreement

HISTORY AND LICENSE

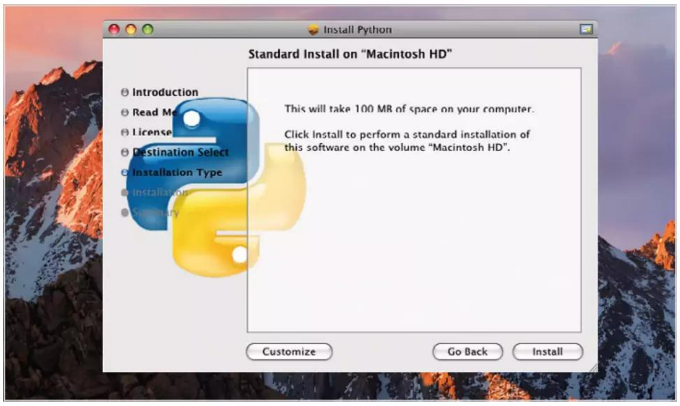
HISTORY OF THE SOFTWARE

Python was created in the early 1990s by Guido van Rossum at Stichting Mathematisch Centrum (CWI, see http://www.cwi.nl) in the Netherlands as a successor of a language called ABC. Guido remains Python's principal author, although it includes many contributions from others.

In 1995, Guido continued his work on Python at the Corporation for National Research Initiatives (CNRI, see http://www.cnri.reston.va.us) in Reston, Virginia where he released several versions of the software.

In May 2000, Guido and the Python core development team moved to BeOpen.com to form the BeOpen PythonLabs team. In October of the same year, the PythonLabs team moved to Digital Creations (now Zope Corporation, see http://www.zope.com). In 2001, the Python Software Foundation (PSF, see http://www.python.org/psf/) was formed, a non-profit organization created specifically to own Python-related Intellectual Property. Zope Corporation is a sponsoring member of the PSF.

STEP 6 Finally you're be presented with the amount of space Python will take up on your system and an Install button, which you need to click to start the actual installation of Python 3.x on to your Mac. You may need to enter your password to authenticate the installation process.

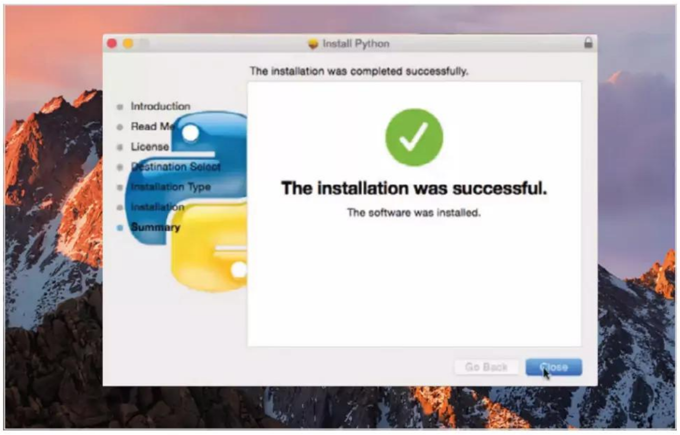

Install Python
Introduction
Read Me
License
Destination Select
Installation Type
Installation
Summary
Customize
Go Back
Install

Standard Install on "Macintosh HD"

This will take 100 MB of space on your computer.

Click Install to perform a standard installation of this software on the volume "Macintosh HD".

STEP 7 The installation shouldn't take too long; the older Mac Mini we used in this section is a little slower than more modern Mac machines and it only took around thirty seconds for the Installation Successful prompt to be displayed.

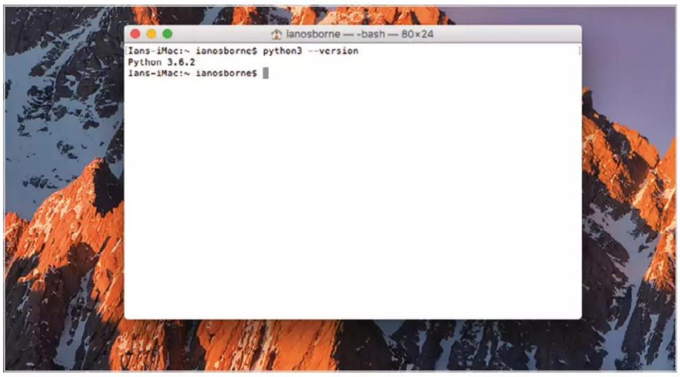

Install Python
Introduction
Read Me
License
Destination Select
Installation Type
Installation
Summary
Go Back
Close

The installation was completed successfully.

The installation was successful.

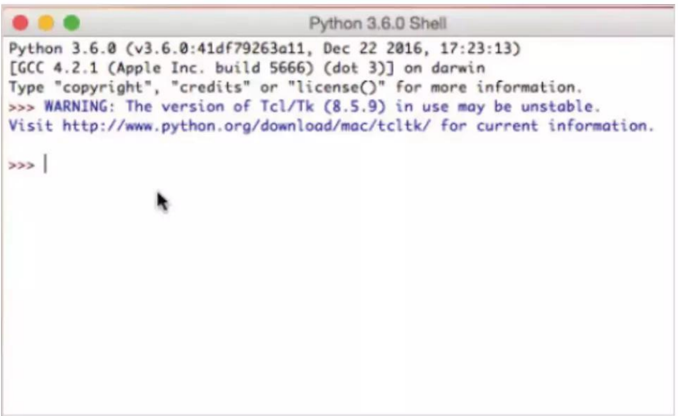
The software was installed.

STEP 8 There's nothing much else left to do in the Python installation wizard so you can click the Close button. If you now drop back into a Terminal session and re-enter the command: `python3 --version`, you can see the new version is now listed. To enter the command line version of Python, you need to enter: `python3`. To exit, it's: `exit()`.


Ianosborne — -bash — 80×24

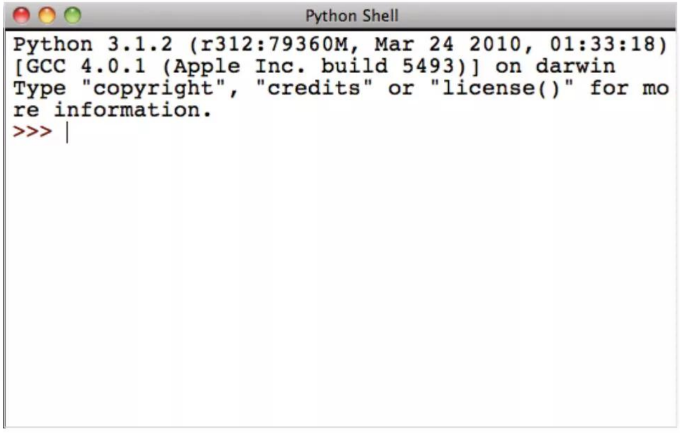
```
Ians-iMac:~ ianosborne$ python3 --version
Python 3.6.2
Ians-iMac:~ ianosborne$ 
```

STEP 9 You need to search in Finder for the Python IDLE; when you've found it, click it to launch and it should look similar to that of the Windows IDLE version shown on the previous page. The only difference being the Mac detected hardware platform it's running on.


Python 3.6.0 Shell

```
Python 3.6.0 (v3.6.0:41df79263a11, Dec 22 2016, 17:23:13)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.9) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.

>>> |
```

STEP 10 Older Mac versions may have trouble with the newer versions of Python, in which case you will need to revert to a previous Python 3.x build; as long as you're using Python 3.x, the code in this book will work for you.


Python Shell

```
Python 3.1.2 (r312:79360M, Mar 24 2010, 01:33:18)
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "copyright", "credits" or "license()" for mo
re information.
>>> |
```

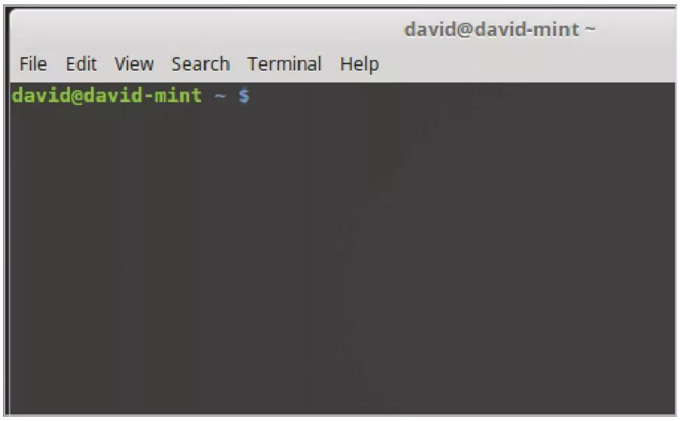
How to Set Up Python in Linux

Python version 2.x is already installed in most Linux distributions but as we're going to be using Python 3.x, there's a little work we need to do first to get hold of it. Thankfully, it's not too difficult.

PYTHON PENGUIN

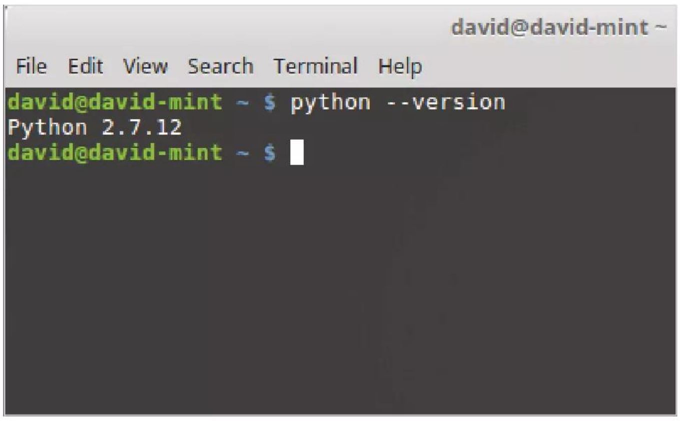
Linux is such a versatile operating system that it's often difficult to nail down just one way of doing something. Different distributions go about installing software in different ways, so we will stick to Linux Mint 18.1 for this particular tutorial.

STEP 1 First you need to ascertain which version of Python is currently installed in your Linux system; as we mentioned, we're going to be using Linux Mint 18.1 for this section. As with macOS, drop into a Terminal by pressing Ctrl+Atl+T.



```
david@david-mint ~ $
```

STEP 2 Next enter: `python --version` into the Terminal screen. You should have the output relating to version 2.x of Python in the display. Ours in this particular case is Python 2.7.12.

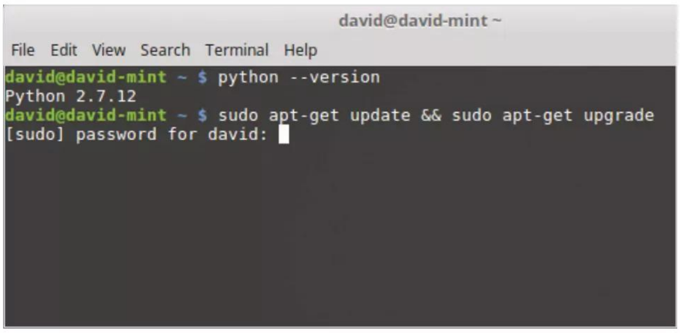


```
david@david-mint ~ $ python --version
Python 2.7.12
david@david-mint ~ $
```

STEP 3 Some Linux distros will automatically update the installation of Python to the latest versions whenever the system is updated. To check, first do a system update and upgrade with:

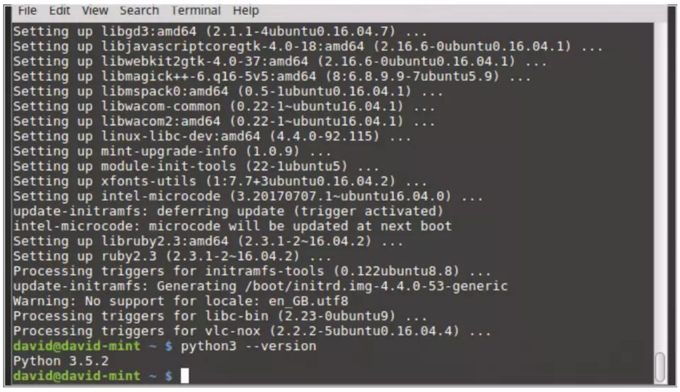
```
sudo apt-get update && sudo apt-get upgrade
```

Enter your password and let the system do any updates.



```
david@david-mint ~ $ python --version
Python 2.7.12
david@david-mint ~ $ sudo apt-get update && sudo apt-get upgrade
[sudo] password for david:
```

STEP 4 Once the update and upgrade is complete, you may need to answer 'Y' to authorise any upgrades, enter: `python3 --version` to see if Python 3.x is updated or even installed. In the case of Linux Mint, the version we have is Python 3.5.2, which is fine for our purposes.

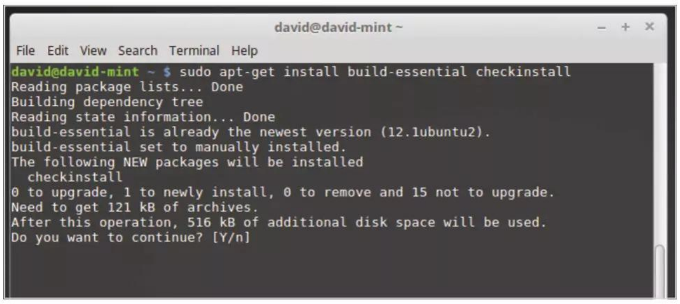


```
Setting up libgd3:amd64 (2.1.1-4ubuntu0.16.04.7) ...
Setting up libjavascriptcoregtk-4.0-18:amd64 (2.16.6-0ubuntu0.16.04.1) ...
Setting up libwebkit2gtk-4.0-37:amd64 (2.16.6-0ubuntu0.16.04.1) ...
Setting up libmagick++-6.q16-5v5:amd64 (8:6.8.9.9-7ubuntu5.9) ...
Setting up libmspack0:amd64 (0.5-1ubuntu0.16.04.1) ...
Setting up libwacom-common (0.22-1~ubuntu16.04.1) ...
Setting up libwacom2:amd64 (0.22-1~ubuntu16.04.1) ...
Setting up linux-libc-dev:amd64 (4.4.0-92.115) ...
Setting up mint-upgrade-info (1.0.9) ...
Setting up module-init-tools (22-1ubuntu5) ...
Setting up xfonts-utils (1:7.7+3ubuntu0.16.04.2) ...
Setting up intel-microcode (3.20170707.1~ubuntu16.04.0) ...
update-initramfs: deferring update (trigger activated)
intel-microcode: microcode will be updated at next boot
Setting up libruby2.3:amd64 (2.3.1-2~16.04.2) ...
Setting up ruby2.3 (2.3.1-2~16.04.2) ...
Processing triggers for initramfs-tools (0.122ubuntu8.8) ...
update-initramfs: Generating /boot/initrd.img-4.4.0-53-generic
Warning: No support for locale: en_GB.utf8
Processing triggers for libc-bin (2.23-0ubuntu9) ...
Processing triggers for vlc-nox (2.2.2-5ubuntu0.16.04.4) ...
david@david-mint ~ $ python3 --version
Python 3.5.2
david@david-mint ~ $
```

STEP 5

However, if you want the latest version, 3.6.2 as per the Python website at the time of writing, you need to build Python from source. Start by entering these commands into the Terminal:

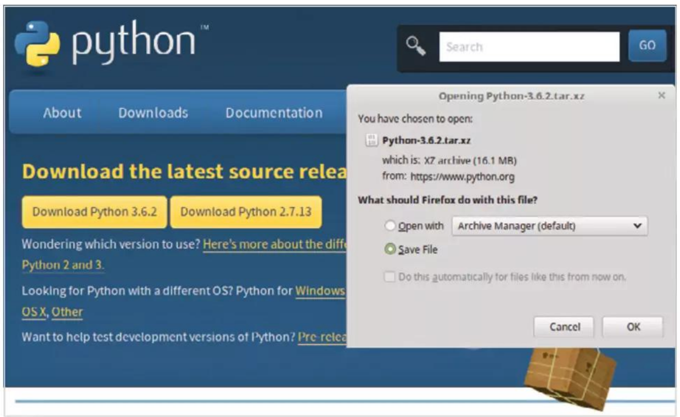
```
sudo apt-get install build-essential checkinstall
sudo apt-get install libreadline-gplv2-dev
libncursesw5-dev libssl-dev libsqlite3-dev tk-dev
libgdbm-dev libc6-dev libbz2-dev
```



```
david@david-mint ~ $ sudo apt-get install build-essential checkinstall
Reading package lists... Done
Building dependency tree
Reading state information... Done
build-essential is already the newest version (12.1ubuntu2).
build-essential set to manually installed.
The following NEW packages will be installed
  checkinstall
0 to upgrade, 1 to newly install, 0 to remove and 15 not to upgrade.
Need to get 121 kB of archives.
After this operation, 516 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

STEP 6

Open up your Linux web browser and go to the Python download page: **www.python.org/downloads**. Click on the Download Python 3.6.2 (or whichever version it's on when you look) to download the source Python-3.6.2.tar.xz file.

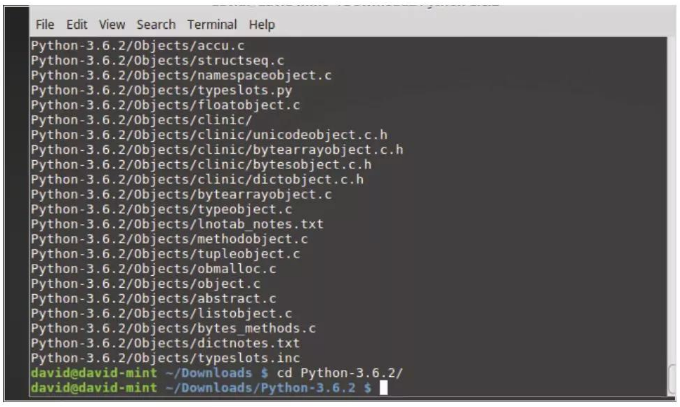


python™
Search
GO
About
Downloads
Documentation
Download the latest source relea
Download Python 3.6.2
Download Python 2.7.13
Wondering which version to use? Here's more about the diff
Python 2 and 3.
Looking for Python with a different OS? Python for Windows
OS X, Other
Want to help test development versions of Python? Pre-relea
Opening Python-3.6.2.tar.xz
Cancel
OK

You have chosen to open:
Python-3.6.2.tar.xz
which is: XZ archive (16.1 MB)
from: https://www.python.org
What should Firefox do with this file?
○ Open with Archive Manager (default)
● Save File
☐ Do this automatically for files like this from now on.

STEP 7

In the Terminal, go the Downloads folder by entering: `cd Downloads/`. Then unzip the contents of the downloaded Python source code with: `tar -xvf Python-3.6.2.tar.xz`. Now enter the newly unzipped folder with `cd Python-3.6.2/`.



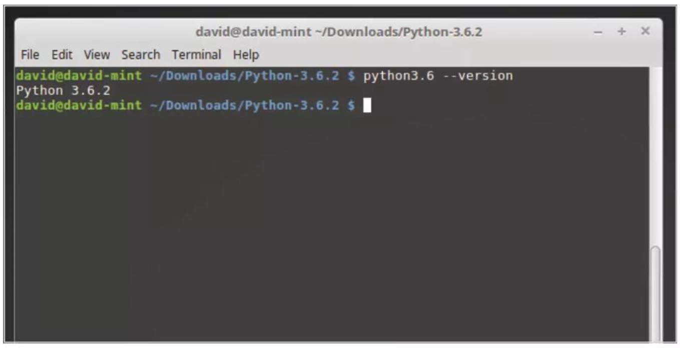
```
Python-3.6.2/Objects/accu.c
Python-3.6.2/Objects/structseq.c
Python-3.6.2/Objects/namespaceobject.c
Python-3.6.2/Objects/typeslots.py
Python-3.6.2/Objects/floatobject.c
Python-3.6.2/Objects/clinic/
Python-3.6.2/Objects/clinic/unicodeobject.c.h
Python-3.6.2/Objects/clinic/bytearrayobject.c.h
Python-3.6.2/Objects/clinic/bytesobject.c.h
Python-3.6.2/Objects/clinic/dictobject.c.h
Python-3.6.2/Objects/bytearrayobject.c
Python-3.6.2/Objects/typeobject.c
Python-3.6.2/Objects/lnotab_notes.txt
Python-3.6.2/Objects/methodobject.c
Python-3.6.2/Objects/tupleobject.c
Python-3.6.2/Objects/obmalloc.c
Python-3.6.2/Objects/object.c
Python-3.6.2/Objects/abstract.c
Python-3.6.2/Objects/listobject.c
Python-3.6.2/Objects/bytes_methods.c
Python-3.6.2/Objects/dictnotes.txt
Python-3.6.2/Objects/typeslots.inc
david@david-mint ~/Downloads $ cd Python-3.6.2/
david@david-mint ~/Downloads/Python-3.6.2 $
```

STEP 8

Within the Python folder, enter:

```
./configure
sudo make altinstall
```

This could a little while depending on the speed of your computer. Once finished, enter: `python3.6 --version` to check the installed latest version.



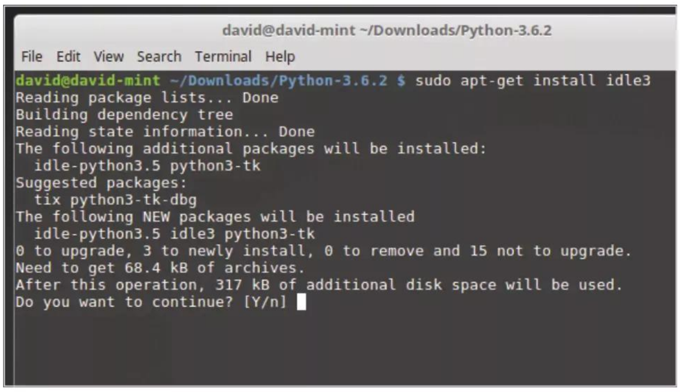
```
david@david-mint ~/Downloads/Python-3.6.2 $ python3.6 --version
Python 3.6.2
david@david-mint ~/Downloads/Python-3.6.2 $
```

STEP 9

For the GUI IDLE, you need to enter the following command into the Terminal:

```
sudo apt-get install idle3
```

The IDLE can then be started with the command: `idle3`. Note, that IDLE runs a different version from the one you installed from source.



```
david@david-mint ~/Downloads/Python-3.6.2 $ sudo apt-get install idle3
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  idle-python3.5 python3-tk
Suggested packages:
  tix python3-tk-dbg
The following NEW packages will be installed
  idle-python3.5 idle3 python3-tk
0 to upgrade, 3 to newly install, 0 to remove and 15 not to upgrade.
Need to get 68.4 kB of archives.
After this operation, 317 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

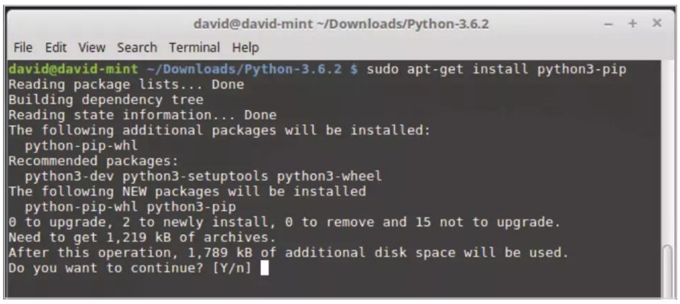
STEP 10

You also need PIP (Pip Installs Packages) which is a tool to help you install more modules and extras. Enter: `sudo apt-get install python3-pip`

PIP is then installed; check for the latest update with:

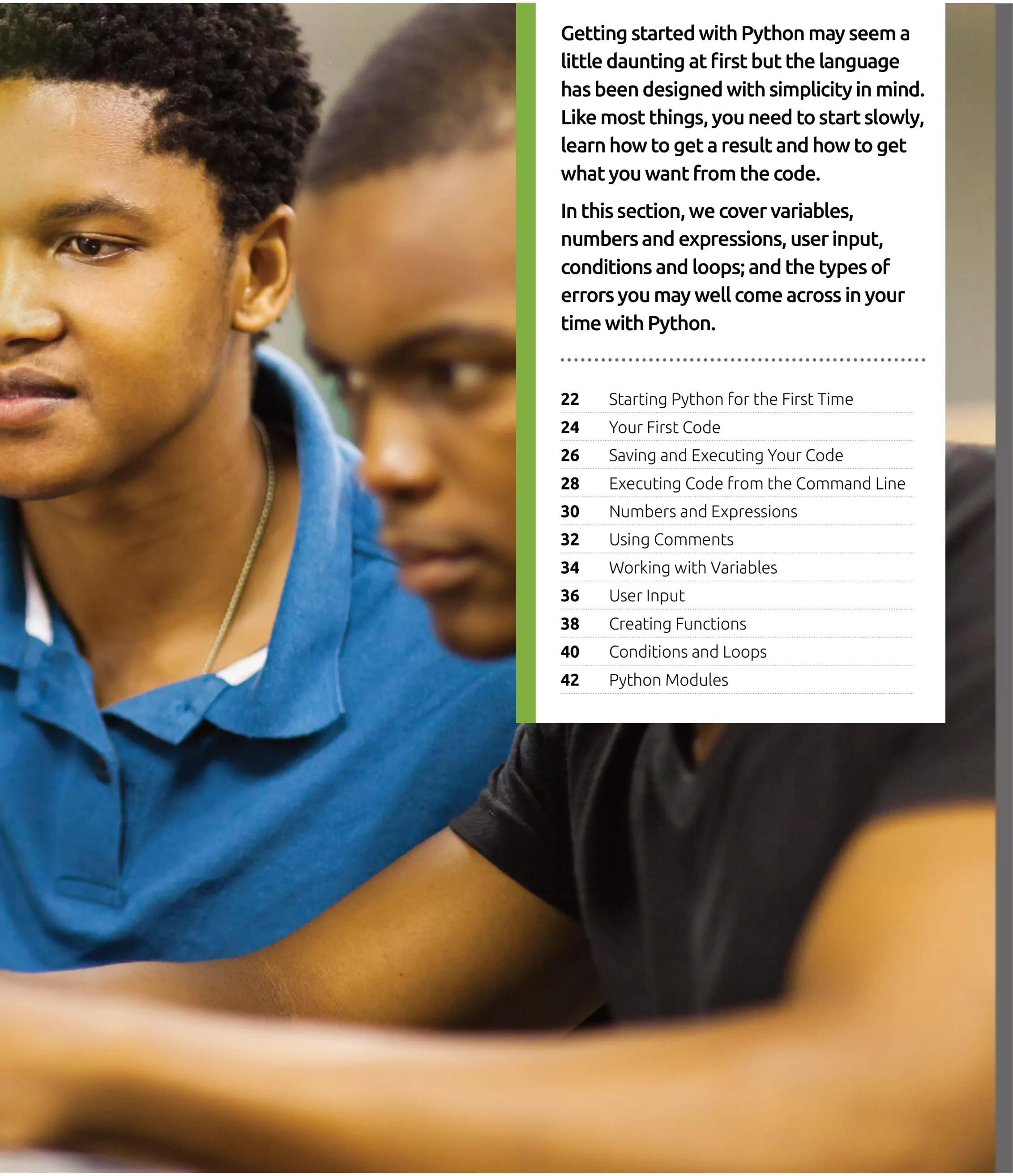
```
pip3 install --upgrade pip
```

When complete, close the Terminal and Python 3.x will be available via the Programming section in your distro's menu.



```
david@david-mint ~/Downloads/Python-3.6.2 $ sudo apt-get install python3-pip
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  python-pip-whl
Recommended packages:
  python3-dev python3-setuptools python3-wheel
The following NEW packages will be installed
  python-pip-whl python3-pip
0 to upgrade, 2 to newly install, 0 to remove and 15 not to upgrade.
Need to get 1,219 kB of archives.
After this operation, 1,789 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Getting Started with Python



Getting started with Python may seem a little daunting at first but the language has been designed with simplicity in mind. Like most things, you need to start slowly, learn how to get a result and how to get what you want from the code.

In this section, we cover variables, numbers and expressions, user input, conditions and loops; and the types of errors you may well come across in your time with Python.

22 Starting Python for the First Time

24 Your First Code

26 Saving and Executing Your Code

28 Executing Code from the Command Line

30 Numbers and Expressions

32 Using Comments

34 Working with Variables

36 User Input

38 Creating Functions

40 Conditions and Loops

42 Python Modules

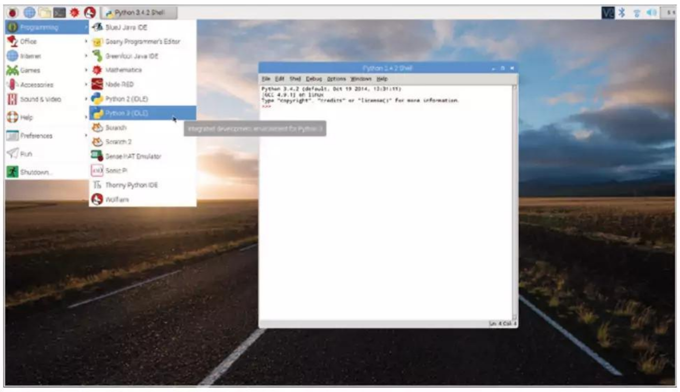
Starting Python for the First Time

We're going to be using the Raspberry Pi as our Python 3 hardware platform. The latest version of Raspbian comes preinstalled with Python 3, version 3.4.2 to be exact, so as long as you have a version 3 Shell, all our code will work.

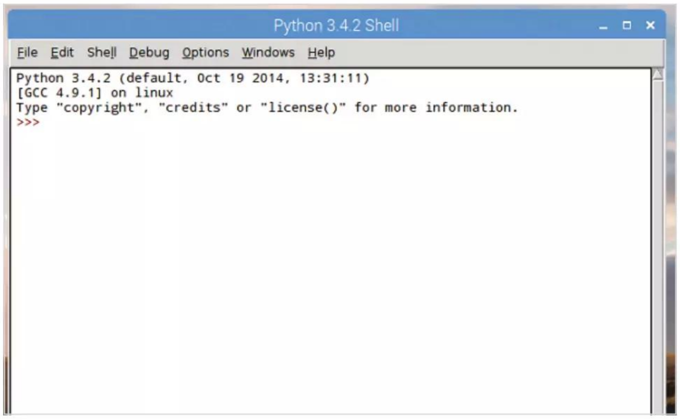
STARTING PYTHON

We're not going to go into the details of getting the Raspberry Pi up and running, there's plenty of material already available on that subject. However, once you're ready, fire up your Pi and get ready for coding.

STEP 1 With the Raspbian desktop loaded, click on the Menu button followed by Programming > Python 3 (IDLE). This opens the Python 3 Shell. Windows and Mac users can find the Python 3 IDLE Shell from within the Windows Start button menu and via Finder.

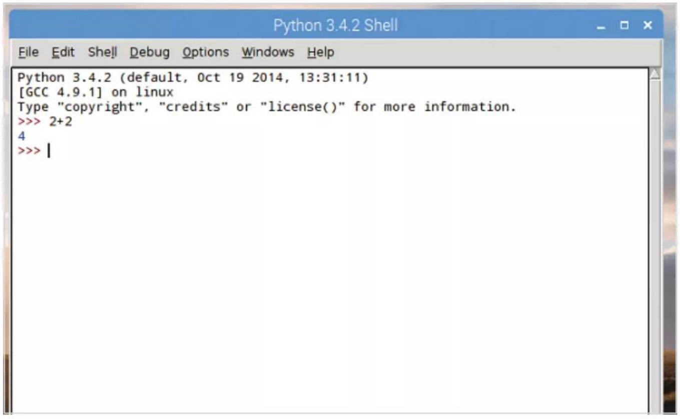

Python 3.4.2 Shell
Programming
Office
Internet
Games
Accessories
Sound & Video
Help
Preferences
Run
Shutdown...
BlueJ Java IDE
Geany Programmer's Editor
Greenfoot Java IDE
Mathematica
Node-RED
Python 2 (IDLE)
Python 3 (IDLE)
Scratch
Scratch 2
Sense HAT Emulator
Sonic Pi
Thonny Python IDE
Wolfram

STEP 2 The Shell is where you can enter code and see the responses and output of code you've programmed into Python. This is a kind of sandbox, where you're able to try out some simple code and processes.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

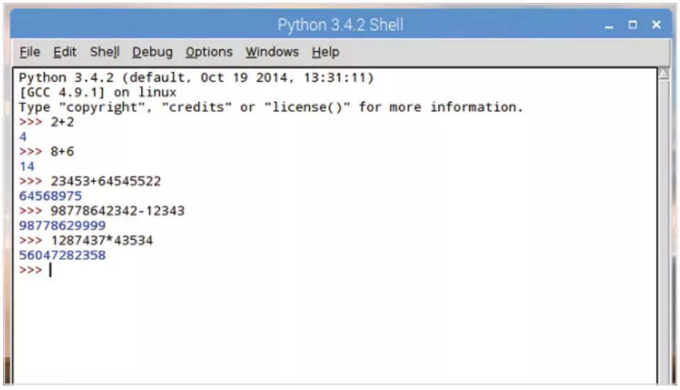
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
```

STEP 3 For example, in the Shell enter: 2+2 After pressing Enter, the next line displays the answer: 4. Basically, Python has taken the 'code' and produced the relevant output.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>> |
```

STEP 4 The Python Shell acts very much like a calculator, since code is basically a series of mathematical interactions with the system. Integers, which are the infinite sequence of whole numbers can easily be added, subtracted, multiplied and so on.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

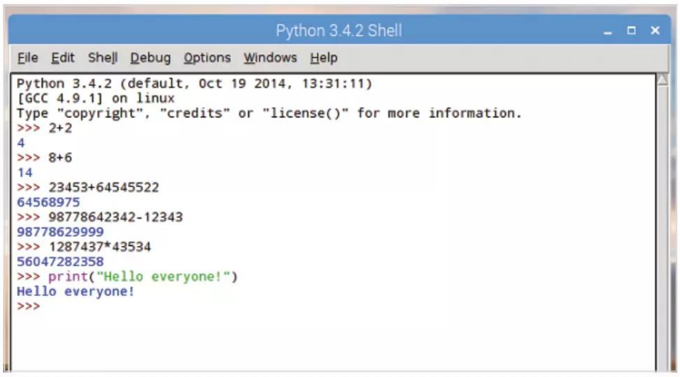
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>> 8+6
14
>>> 23453+64545522
64568975
>>> 98778642342-12343
98778629999
>>> 1287437*43534
56047282358
>>> |
```

STEP 5

While that's very interesting, it's not particularly exciting. Instead, try this:

```
print("Hello everyone!")
```

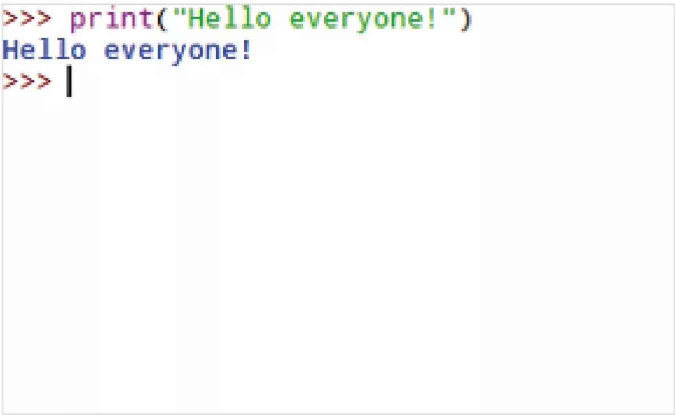
Just like the code we entered in Sublime in the Installing a Text Editor section of this book.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>> 8+6
14
>>> 23453+64545522
64568975
>>> 98778642342-12343
98778629999
>>> 1287437*43534
56047282358
>>> print("Hello everyone!")
Hello everyone!
>>>
```

STEP 6

This is a little more like it, since you've just produced your first bit of code. The Print command is fairly self-explanatory, it prints things. Python 3 requires the brackets as well as quote marks in order to output content to the screen, in this case the 'Hello everyone!' bit.

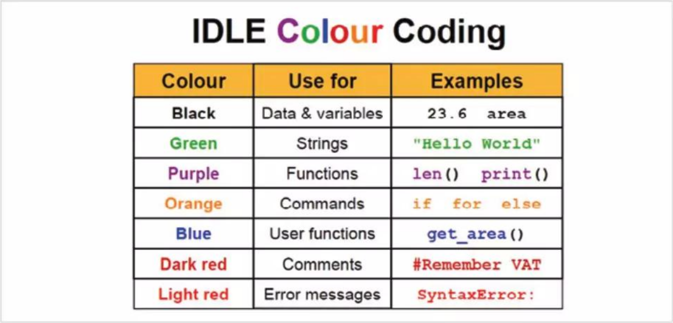


```
>>> print("Hello everyone!")
Hello everyone!
>>> |
```

STEP 7

You may have noticed the colour coding within the Python IDLE. The colours represent different elements of Python code. They are:

Black – Data and Variables
Green – Strings
Purple – Functions
Orange – Commands
Blue – User Functions
Dark Red – Comments
Light Red – Error Messages

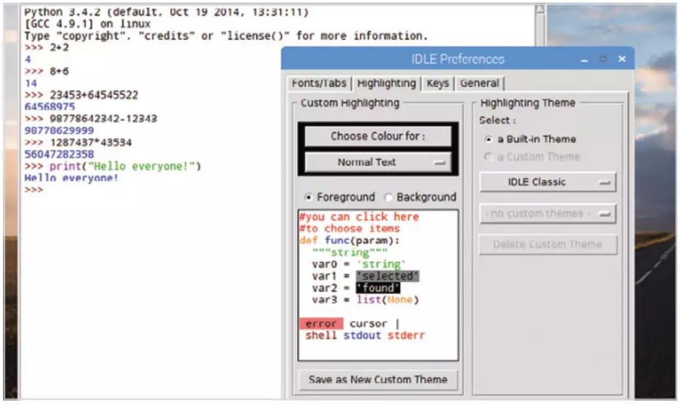


IDLE Colour Coding

Colour	Use for	Examples
Black	Data & variables	`23.6  area`
Green	Strings	`"Hello World"`
Purple	Functions	`len()  print()`
Orange	Commands	`if  for  else`
Blue	User functions	`get_area()`
Dark red	Comments	`#Remember VAT`
Light red	Error messages	`SyntaxError:`

STEP 8

The Python IDLE is a configurable environment. If you don't like the way the colours are represented, then you can always change them via Options > Configure IDLE and clicking on the Highlighting tab. However, we don't recommend that, as you won't be seeing the same as our screenshots.

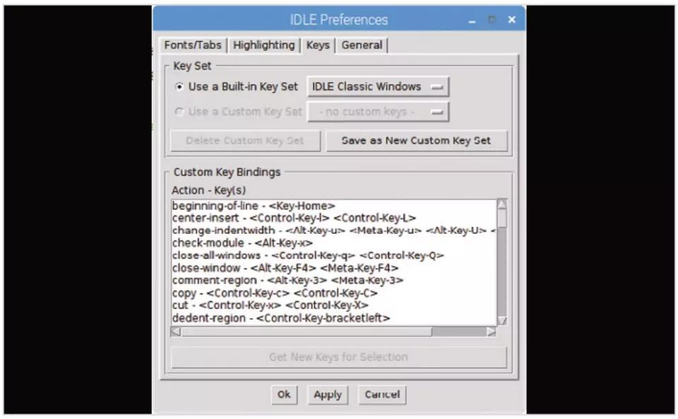

IDLE Preferences
Fonts/Tabs
Highlighting
Keys
General
Custom Highlighting
Choose Colour for :
Normal Text
● Foreground ○ Background
Save as New Custom Theme
Highlighting Theme
Select :
● a Built-in Theme
○ a Custom Theme
IDLE Classic
- no custom themes -
Delete Custom Theme

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>> 8+6
14
>>> 23453+64545522
64568975
>>> 98778642342-12343
98778629999
>>> 1287437*43534
56047282358
>>> print("Hello everyone!")
Hello everyone!
>>>
```

```
#you can click here
#to choose items
def func(param):
  """string"""
  var0 = 'string'
  var1 = 'selected'
  var2 = 'found'
  var3 = list(None)

error cursor |
shell stdout stderr
```

STEP 9

Just like most programs available, regardless of the operating system, there are numerous shortcut keys available. We don't have room for them all here but within the Options > Configure IDLE and under the Keys tab, you can see a list of the current bindings.

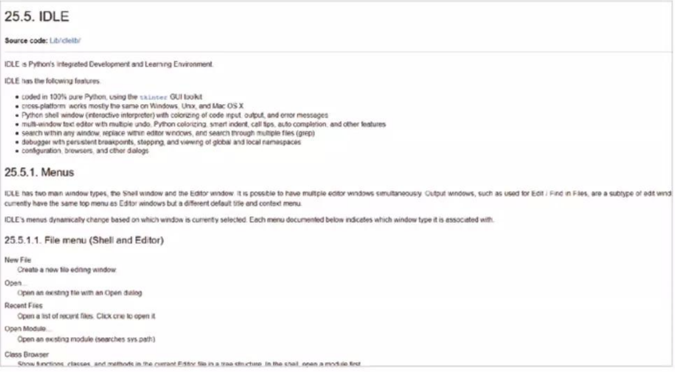

IDLE Preferences
Fonts/Tabs
Highlighting
Keys
General
Key Set
● Use a Built-in Key Set
IDLE Classic Windows
○ Use a Custom Key Set
- no custom keys -
Delete Custom Key Set
Save as New Custom Key Set
Custom Key Bindings
Get New Keys for Selection
Ok
Apply
Cancel

Action - Key(s)

```
beginning-of-line - <Key-Home>
center-insert - <Control-Key-l> <Control-Key-L>
change-indentwidth - <Alt-Key-u> <Meta-Key-u> <Alt-Key-U> -
check-module - <Alt-Key-x>
close-all-windows - <Control-Key-q> <Control-Key-Q>
close-window - <Alt-Key-F4> <Meta-Key-F4>
comment-region - <Alt-Key-3> <Meta-Key-3>
copy - <Control-Key-c> <Control-Key-C>
cut - <Control-Key-x> <Control-Key-X>
dedent-region - <Control-Key-bracketleft>
```

STEP 10

The Python IDLE is a power interface and one that's actually been written in Python using one of the available GUI toolkits. If you want to know the many ins and outs of the Shell, we recommend you take a few moments to view **www.docs.python.org/3/library/idle.html**, which details many of the IDLE's features.



25.5. IDLE

Source code: Lib/idlelib/

IDLE is Python's Integrated Development and Learning Environment.

IDLE has the following features:

- coded in 100% pure Python, using the `tkinter` GUI toolkit
- cross-platform: works mostly the same on Windows, Unix, and Mac OS X
- Python shell window (interactive interpreter) with colorizing of code input, output, and error messages
- multi-window text editor with multiple undo, Python colorizing, smart indent, call tips, auto completion, and other features
- search within any window, replace within editor windows, and search through multiple files (grep)
- debugger with persistent breakpoints, stepping, and viewing of global and local namespaces
- configuration, browsers, and other dialogs

25.5.1. Menus

IDLE has two main window types, the Shell window and the Editor window. It is possible to have multiple editor windows simultaneously. Output windows, such as used for Edit / Find in Files, are a subtype of edit wind
currently have the same top menu as Editor windows but a different default title and context menu.

IDLE's menus dynamically change based on which window is currently selected. Each menu documented below indicates which window type it is associated with.

25.5.1.1. File menu (Shell and Editor)

New File
Create a new file editing window.
Open...
Open an existing file with an Open dialog.
Recent Files
Open a list of recent files. Click one to open it.
Open Module...
Open an existing module (searches sys.path).
Class Browser
Show functions, classes, and methods in the current Editor file in a tree structure. In the shell, open a module first.

Your First Code

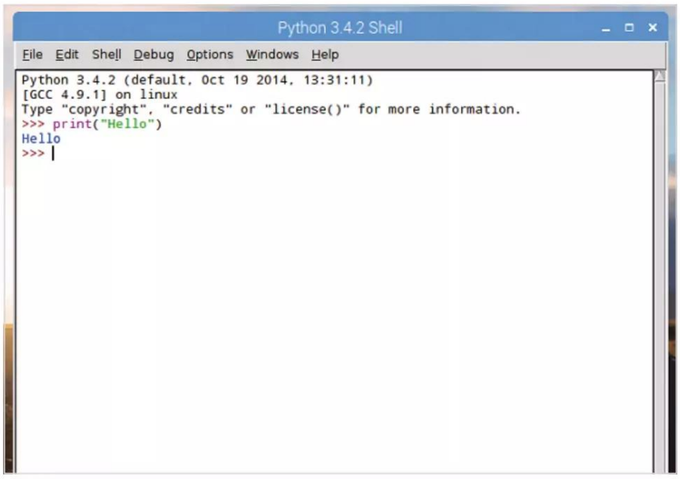
Essentially, you've already written your first piece of code with the 'print("Hello everyone!")' function from the previous tutorial. However, let's expand that and look at entering your code and playing around with some other Python examples.

PLAYING WITH PYTHON

With most languages, computer or human, it's all about remembering and applying the right words to the right situation. You're not born knowing these words, so you need to learn them.

STEP 1 If you've closed Python 3 IDLE, reopen it in whichever operating system version you prefer. In the Shell, enter the familiar following:

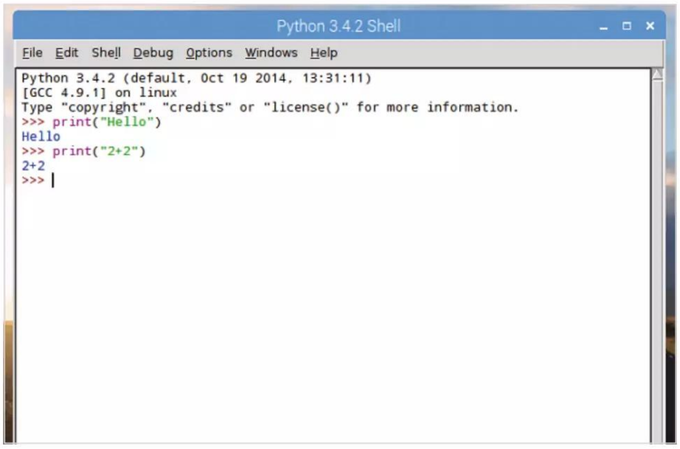
`print("Hello")`


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello")
Hello
>>> |
```

STEP 2 Just as predicted, the word Hello appears in the Shell as blue text, indicating output from a string. It's fairly straightforward and doesn't require too much explanation. Now try:

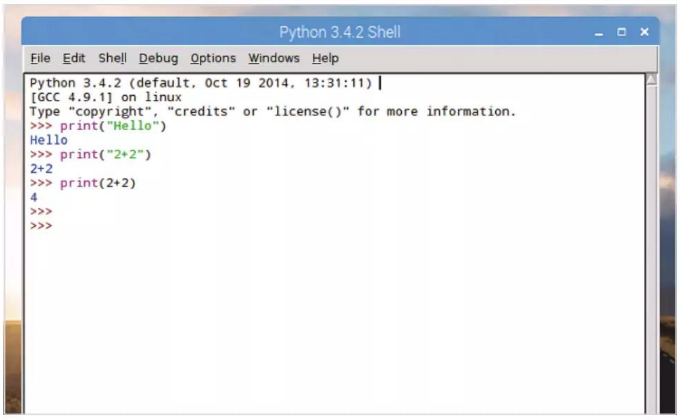
`print("2+2")`


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello")
Hello
>>> print("2+2")
2+2
>>> |
```

STEP 3 You can see that instead of the number 4, the output is the 2+2 you asked to be printed to the screen. The quotation marks are defining what's being outputted to the IDLE Shell; to print the total of 2+2 you need to remove the quotes:

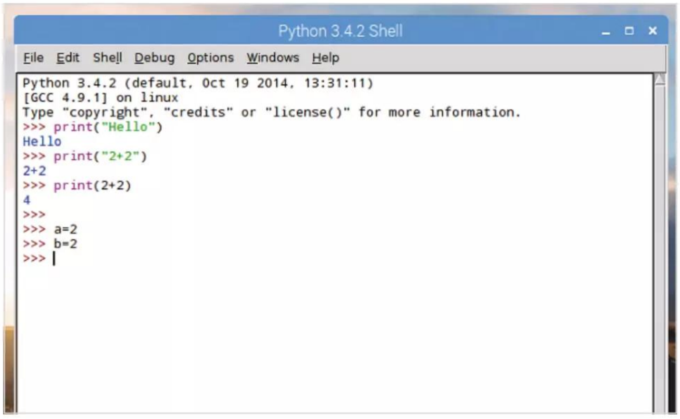
`print(2+2)`


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11) |
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello")
Hello
>>> print("2+2")
2+2
>>> print(2+2)
4
>>>
>>>
```

STEP 4 You can continue as such, printing 2+2, 464+2343 and so on to the Shell. An easier way is to use a variable, which is something we will cover in more depth later. For now, enter:

```
a=2
b=2
```

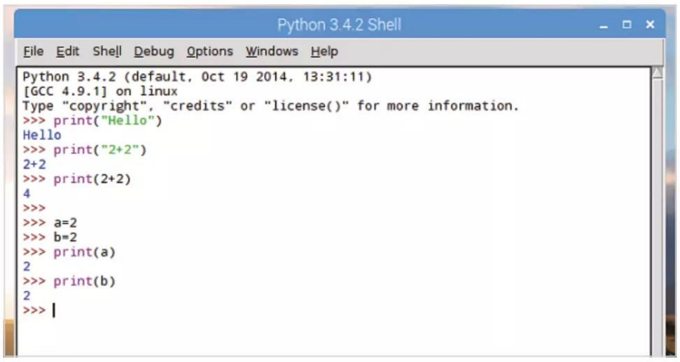

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello")
Hello
>>> print("2+2")
2+2
>>> print(2+2)
4
>>>
>>> a=2
>>> b=2
>>> |
```

STEP 5

What you have done here is assign the letters a and b two values: 2 and 2. These are now variables, which can be called upon by Python to output, add, subtract, divide and so on for as long as their numbers stay the same. Try this:

```
print(a)
print(b)
```



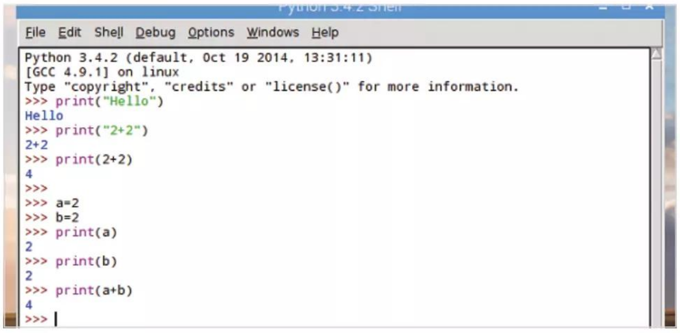
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello")
Hello
>>> print("2+2")
2+2
>>> print(2+2)
4
>>>
>>> a=2
>>> b=2
>>> print(a)
2
>>> print(b)
2
>>> |
```

STEP 6

The output of the last step displays the current values of both a and b individually, as you've asked them to be printed separately. If you want to add them up, you can use the following:

```
print(a+b)
```

This code simply takes the values of a and b, adds them together and outputs the result.

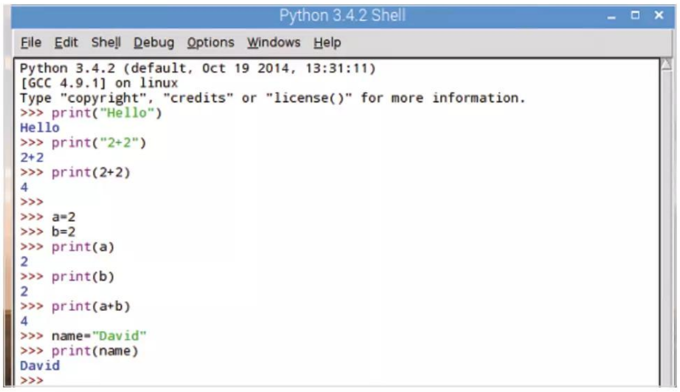


```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello")
Hello
>>> print("2+2")
2+2
>>> print(2+2)
4
>>>
>>> a=2
>>> b=2
>>> print(a)
2
>>> print(b)
2
>>> print(a+b)
4
>>> |
```

STEP 7

You can play around with different kinds of variables and the Print function. For example, you could assign variables for someone's name:

```
name=”David”
print(name)
```



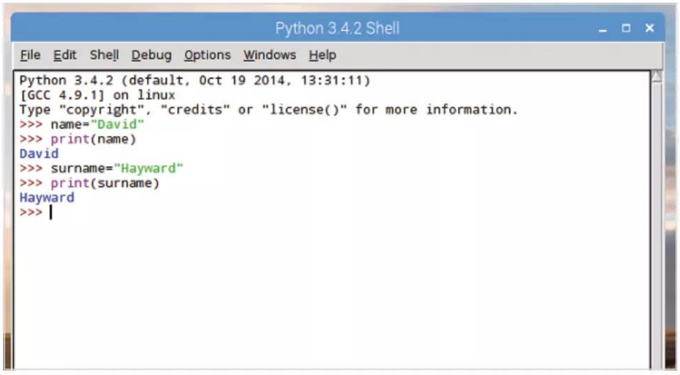
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello")
Hello
>>> print("2+2")
2+2
>>> print(2+2)
4
>>>
>>> a=2
>>> b=2
>>> print(a)
2
>>> print(b)
2
>>> print(a+b)
4
>>> name="David"
>>> print(name)
David
>>>
```

STEP 8

Now let's add a surname:

```
surname=”Hayward”
print(surname)
```

You now have two variables containing a first name and a surname and you can print them independently.



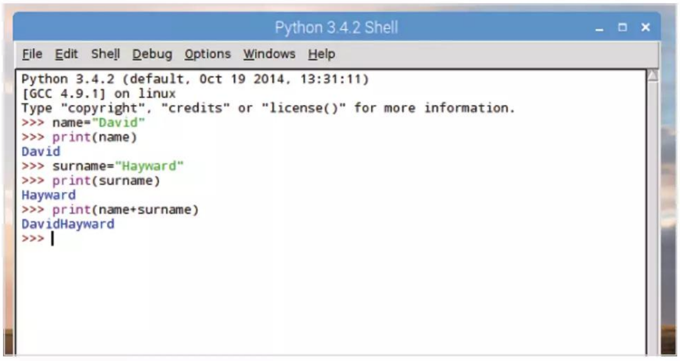
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name="David"
>>> print(name)
David
>>> surname="Hayward"
>>> print(surname)
Hayward
>>> |
```

STEP 9

If we were to apply the same routine as before, using the + symbol, the name wouldn't appear correctly in the output in the Shell. Try it:

```
print(name+surname)
```

You need a space between the two, defining them as two separate values and not something you mathematically play around with.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name="David"
>>> print(name)
David
>>> surname="Hayward"
>>> print(surname)
Hayward
>>> print(name+surname)
DavidHayward
>>> |
```

STEP 10

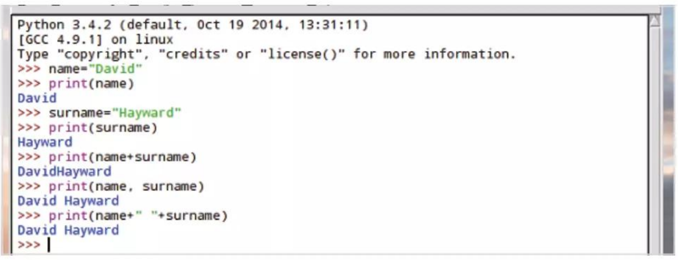
In Python 3 you can separate the two variables with a space using a comma:

```
print(name, surname)
```

Alternatively, you can add the space yourself:

```
print(name+” “+surname)
```

The use of the comma is much neater, as you can see. Congratulations, you've just taken your first steps into the wide world of Python.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name="David"
>>> print(name)
David
>>> surname="Hayward"
>>> print(surname)
Hayward
>>> print(name+surname)
DavidHayward
>>> print(name, surname)
David Hayward
>>> print(name+" "+surname)
David Hayward
>>> |
```

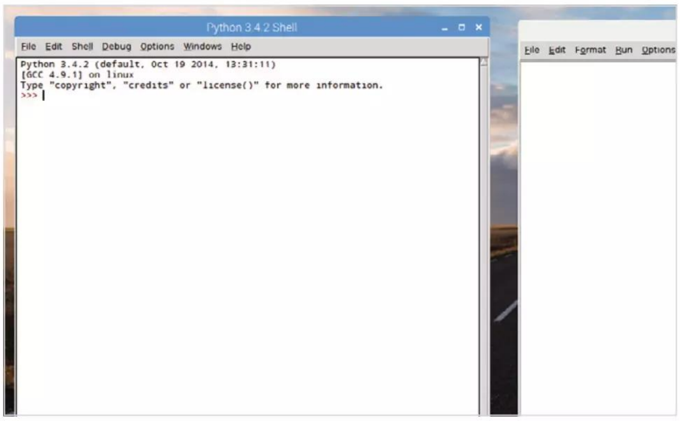
Saving and Executing Your Code

Whilst working in the IDLE Shell is perfectly fine for small code snippets, it's not designed for entering longer program listings. In this section you're going to be introduced to the IDLE Editor, where you will be working from now on.

EDITING CODE

You will eventually reach a point where you have to move on from inputting single lines of code into the Shell. Instead, the IDLE Editor will allow you to save and execute your Python code.

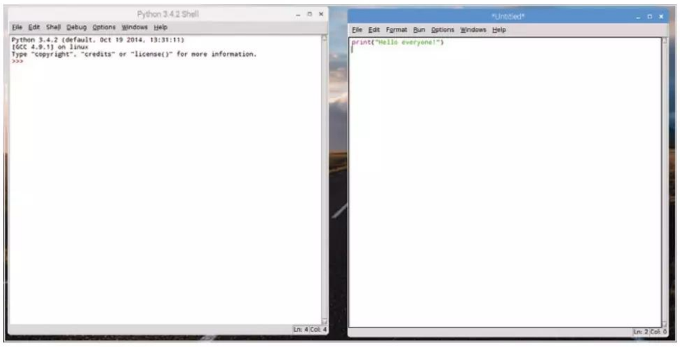
STEP 1 First, open the Python IDLE Shell and when it's up, click on File > New File. This will open a new window with Untitled as its name. This is the Python IDLE Editor and within it you can enter the code needed to create your future programs.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
File Edit Format Run Options

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> 
```

STEP 2 The IDLE Editor is, for all intents and purposes, a simple text editor with Python features, colour coding and so on; much in the same vein as Sublime. You enter code as you would within the Shell, so taking an example from the previous tutorial, enter:

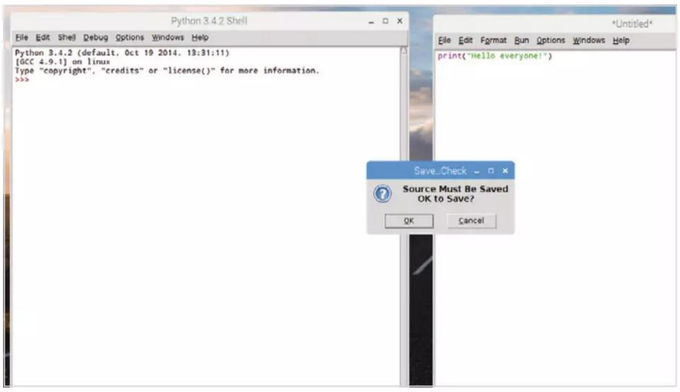
```
print("Hello everyone!")
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Untitled
File Edit Format Run Options Windows Help
Ln: 4 Col: 4
Ln: 2 Col: 0

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
```

```
print("Hello everyone!")
```

STEP 3 You can see that the same colour coding is in place in the IDLE Editor as it is in the Shell, enabling you to better understand what's going on with your code. However, to execute the code you need to first save it. Press F5 and you get a Save…Check box open.

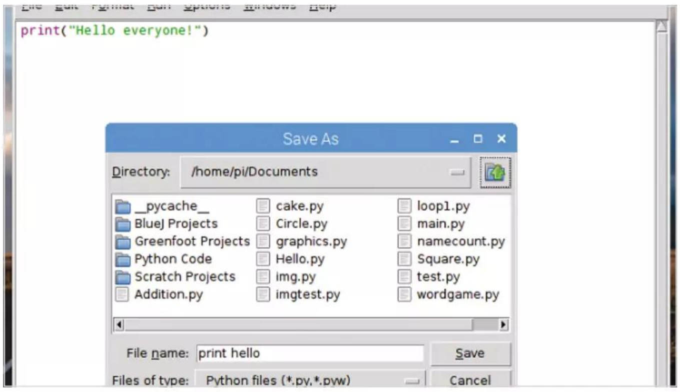

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Untitled
File Edit Format Run Options Windows Help
Save...Check
OK
Cancel

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
```

```
print("Hello everyone!")
```

Source Must Be Saved
OK to Save?

STEP 4 Click on the OK button in the Save box and select a destination where you'll save all your Python code. The destination can be a dedicated folder called Python or you can just dump it wherever you like. Remember to keep a tidy drive though, to help you out in the future.


Save As
Save
Cancel

```
print("Hello everyone!")
```

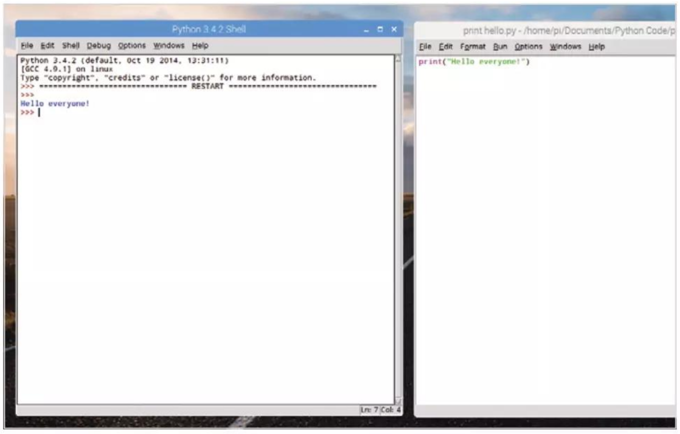
Directory: /home/pi/Documents

__pycache__
BlueJ Projects
Greenfoot Projects
Python Code
Scratch Projects
Addition.py
cake.py
Circle.py
graphics.py
Hello.py
img.py
imgtest.py
loop1.py
main.py
namecount.py
Square.py
test.py
wordgame.py

File name: print hello

Files of type: Python files (*.py,*.pyw)

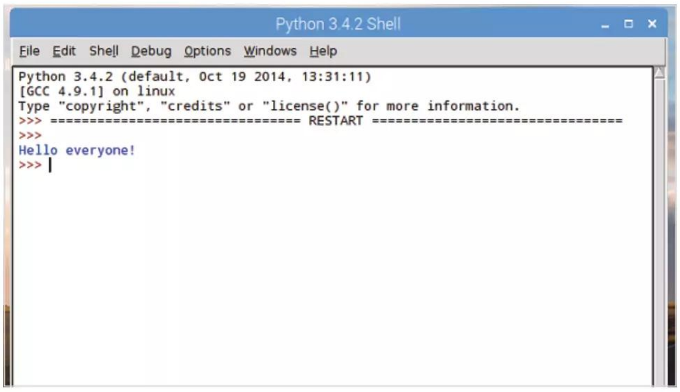
STEP 5 Enter a name for your code, 'print hello' for example, and click on the Save button. Once the Python code is saved it's executed and the output will be detailed in the IDLE Shell. In this case, the words 'Hello everyone!'.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
print hello.py - /home/pi/Documents/Python Code/p
File Edit Format Run Options Windows Help
Ln: 7 Col: 4

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Hello everyone!
>>> |
```

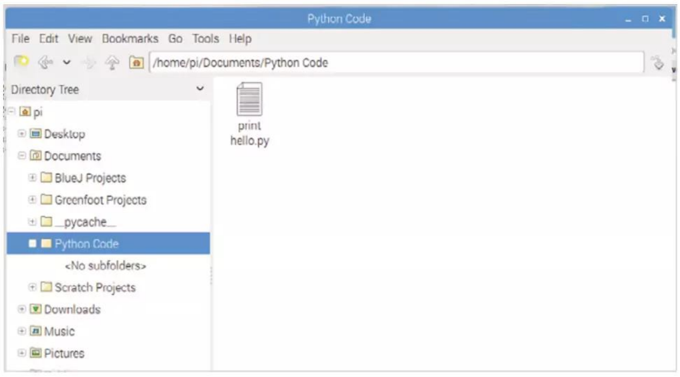
```
print("Hello everyone!")
```

STEP 6 This is how the vast majority of your Python code will be conducted. Enter it into the Editor, hit F5, save the code and look at the output in the Shell. Sometimes things will differ, depending on whether you've requested a separate window, but essentially that's the process. It's the process we will use throughout this book, unless otherwise stated.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Hello everyone!
>>> |
```

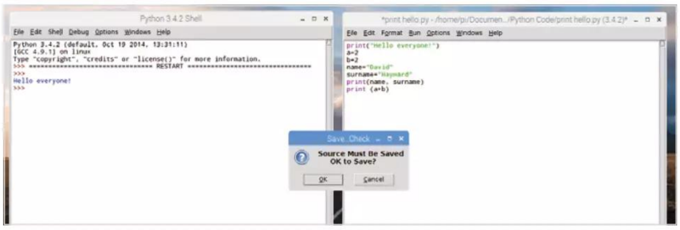
STEP 7 If you open the file location of the saved Python code, you can see that it ends in a .py extension. This is the default Python file name. Any code you create will be whatever.py and any code downloaded from the many Internet Python resource sites will be .py. Just ensure that the code is written for Python 3.


Python Code
File Edit View Bookmarks Go Tools Help
/home/pi/Documents/Python Code
Directory Tree
pi
Desktop
Documents
BlueJ Projects
Greenfoot Projects
__pycache__
Python Code
<No subfolders>
Scratch Projects
Downloads
Music
Pictures
print
hello.py

STEP 8 Let's extend the code and enter a few examples from the previous tutorial:

```
a=2
b=2
name=”David”
surname=”Hayward”
print(name, surname)
print (a+b)
```

If you press F5 now you'll be asked to save the file, again, as it's been modified from before.

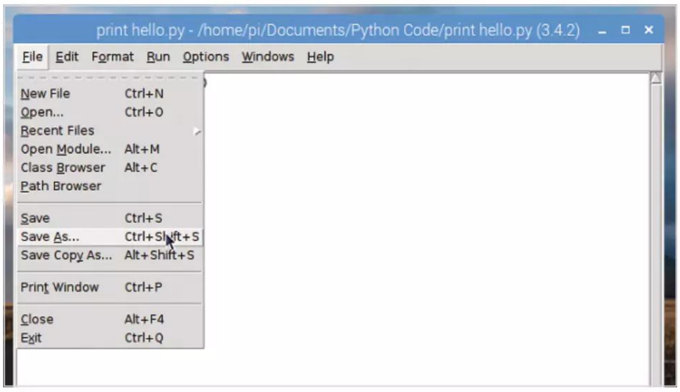

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
print hello.py - /home/pi/Documen.../Python Code/print hello.py (3.4.2)
File Edit Format Run Options Windows Help
Save Check
OK
Cancel

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Hello everyone!
>>>
```

```
print("Hello everyone!")
a=2
b=2
name="David"
surname="Hayward"
print(name, surname)
print (a+b)
```

Source Must Be Saved
OK to Save?

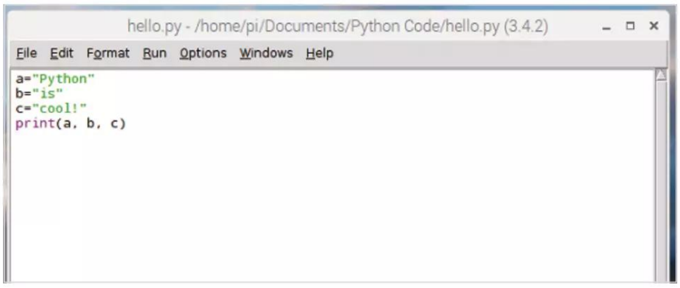
STEP 9 If you click the OK button, the file will be overwritten with the new code entries, and executed, with the output in the Shell. It's not a problem with just these few lines but if you were to edit a larger file, overwriting can become an issue. Instead, use File > Save As from within the Editor to create a backup.


print hello.py - /home/pi/Documents/Python Code/print hello.py (3.4.2)
File Edit Format Run Options Windows Help
New File Ctrl+N
Open... Ctrl+O
Recent Files
Open Module... Alt+M
Class Browser Alt+C
Path Browser
Save Ctrl+S
Save As... Ctrl+Shift+S
Save Copy As... Alt+Shift+S
Print Window Ctrl+P
Close Alt+F4
Exit Ctrl+Q

STEP 10 Now create a new file. Close the Editor, and open a new instance (File > New File from the Shell). Enter the following and save it as hello.py:

```
a=”Python”
b=”is”
c=”cool!”
print(a, b, c)
```

You will use this code in the next tutorial.


hello.py - /home/pi/Documents/Python Code/hello.py (3.4.2)
File Edit Format Run Options Windows Help

```
a="Python"
b="is"
c="cool!"
print(a, b, c)
```

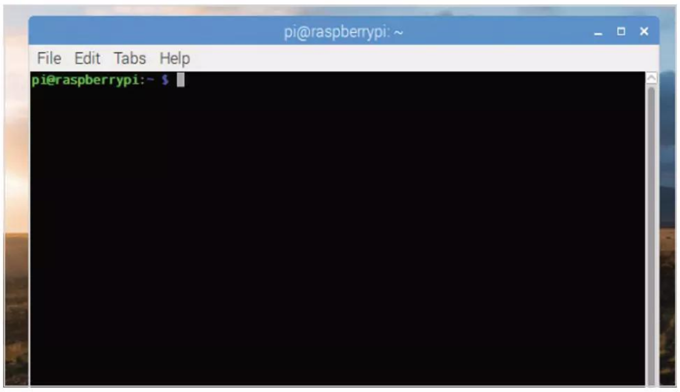
Executing Code from the Command Line

Although we're working from the GUI IDLE throughout this book, it's worth taking a look at Python's command line handling. We already know there's a command line version of Python but it's also used to execute code.

COMMAND THE CODE

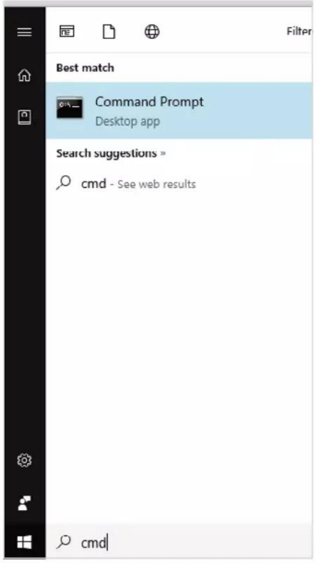
Using the code we created in the previous tutorial, the one we named hello.py, let's see how you can run code that was made in the GUI at the command line level.

STEP 1 Python, in Linux, comes with two possible ways of executing code via the command line. One of the ways is with Python 2, whilst the other uses the Python 3 libraries and so on. First though, drop into the command line or Terminal on your operating system.

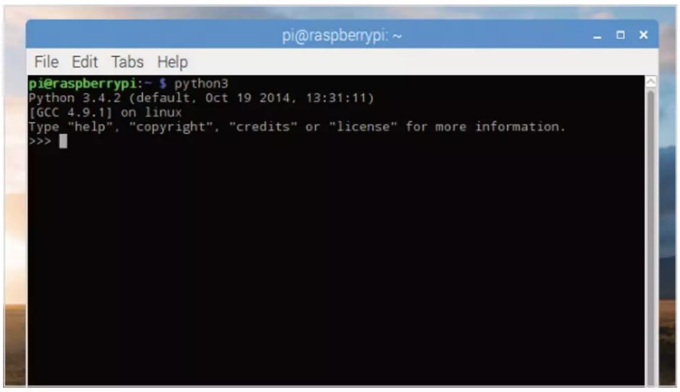

pi@raspberrypi: ~
File Edit Tabs Help

```
pi@raspberrypi:~ $ 
```

STEP 2 Just as before, we're using a Raspberry Pi: Windows users will need to click the Start button and search for CMD, then click the Command Line returned search; and macOS users can get access to their command line by clicking Go > Utilities > Terminal.


Filter
Best match
Command Prompt
Desktop app
Search suggestions
cmd - See web results
cmd

STEP 3 Now you're at the command line we can start Python. For Python 3 you need to enter the command **python3** and press Enter. This will put you into the command line version of the Shell, with the familiar three right-facing arrows as the cursor (>>>).

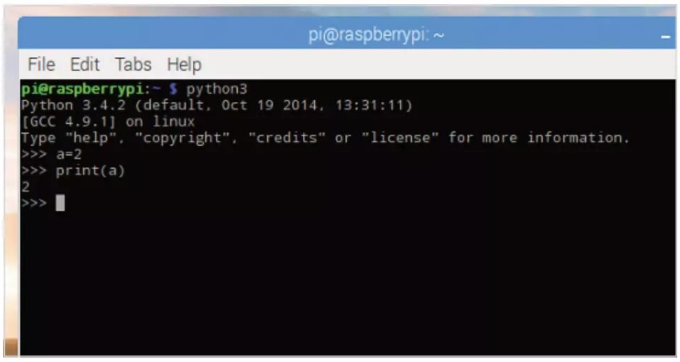

pi@raspberrypi: ~
File Edit Tabs Help

```
pi@raspberrypi:~ $ python3
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

STEP 4 From here you're able to enter the code you've looked at previously, such as:

```
a=2
print(a)
```

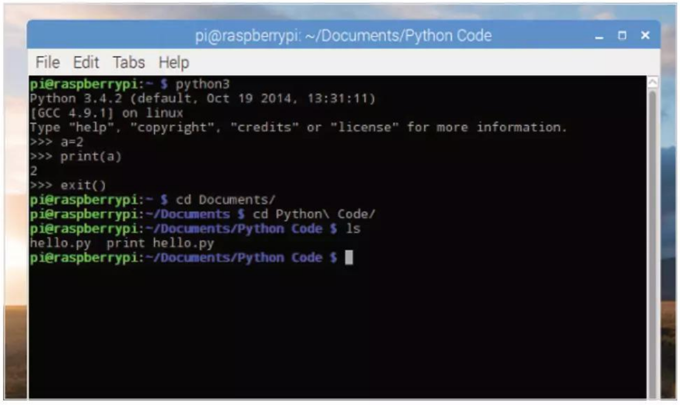
You can see that it works exactly the same.


pi@raspberrypi: ~
File Edit Tabs Help

```
pi@raspberrypi:~ $ python3
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a=2
>>> print(a)
2
>>> 
```

STEP 5

Now enter: **exit()** to leave the command line Python session and return you back to the command prompt. Enter the folder where you saved the code from the previous tutorial and list the available files within; hopefully you should see the hello.py file.


pi@raspberrypi: ~/Documents/Python Code
File Edit Tabs Help

```
pi@raspberrypi:~ $ python3
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a=2
>>> print(a)
2
>>> exit()
pi@raspberrypi:~ $ cd Documents/
pi@raspberrypi:~/Documents $ cd Python\ Code/
pi@raspberrypi:~/Documents/Python Code $ ls
hello.py  print hello.py
pi@raspberrypi:~/Documents/Python Code $ 
```

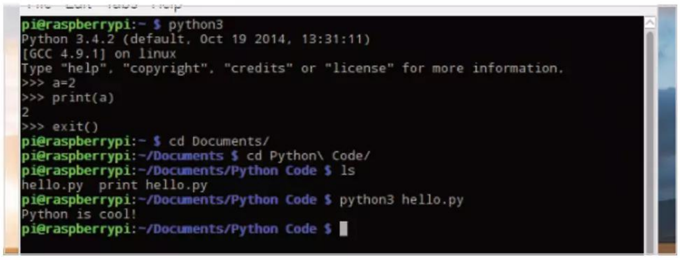
STEP 6

From within the same folder as the code you're going to run, enter the following into the command line:

```
python3 hello.py
```

This will execute the code we created, which to remind you is:

```
a=”Python”
b=”is”
c=”cool!”
print(a, b, c)
```

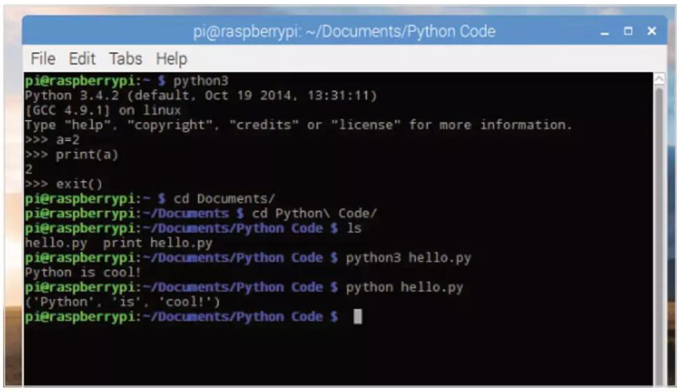


```
pi@raspberrypi:~ $ python3
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a=2
>>> print(a)
2
>>> exit()
pi@raspberrypi:~ $ cd Documents/
pi@raspberrypi:~/Documents $ cd Python\ Code/
pi@raspberrypi:~/Documents/Python Code $ ls
hello.py  print hello.py
pi@raspberrypi:~/Documents/Python Code $ python3 hello.py
Python is cool!
pi@raspberrypi:~/Documents/Python Code $ 
```

STEP 7

Naturally, since this is Python 3 code, using the syntax and layout that's unique to Python 3, it only works when you use the python3 command. If you like, try the same with Python 2 by entering:

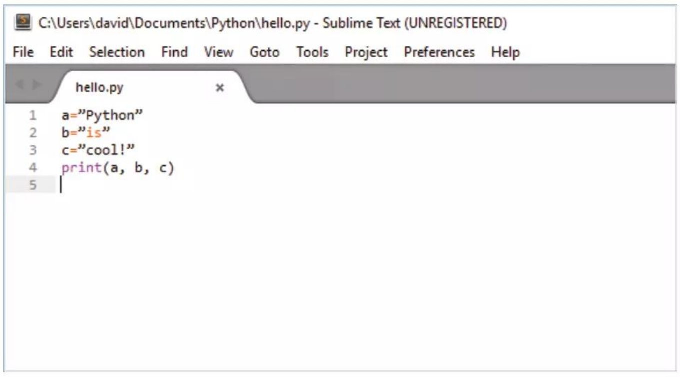
```
python hello.py
```


pi@raspberrypi: ~/Documents/Python Code
File Edit Tabs Help

```
pi@raspberrypi:~ $ python3
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a=2
>>> print(a)
2
>>> exit()
pi@raspberrypi:~ $ cd Documents/
pi@raspberrypi:~/Documents $ cd Python\ Code/
pi@raspberrypi:~/Documents/Python Code $ ls
hello.py  print hello.py
pi@raspberrypi:~/Documents/Python Code $ python3 hello.py
Python is cool!
pi@raspberrypi:~/Documents/Python Code $ python hello.py
('Python', 'is', 'cool!')
pi@raspberrypi:~/Documents/Python Code $ 
```

STEP 8

The result of running Python 3 code from the Python 2 command line is quite obvious. Whilst it doesn't error out in any way, due to the differences between the way Python 3 handles the Print command over Python 2, the result isn't as we expected. Using Sublime for the moment, open the hello.py file.

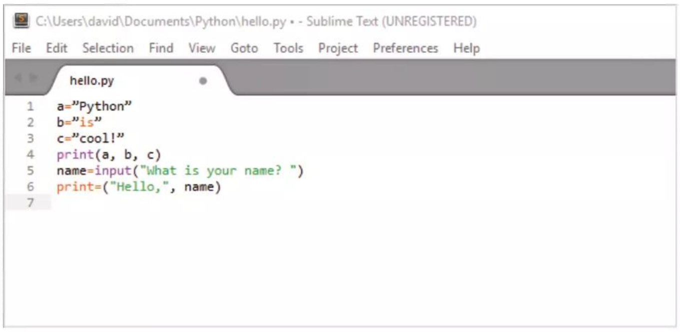

C:\Users\david\Documents\Python\hello.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
hello.py

```
a=”Python”
b=”is”
c=”cool!”
print(a, b, c)
```

STEP 9

Since Sublime Text isn't available for the Raspberry Pi, you're going to temporarily leave the Pi for the moment and use Sublime as an example that you don't necessarily need to use the Python IDLE. With the hello.py file open, alter it to include the following:

```
name=input(“What is your name? “)
print(“Hello,”, name)
```


C:\Users\david\Documents\Python\hello.py • - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
hello.py

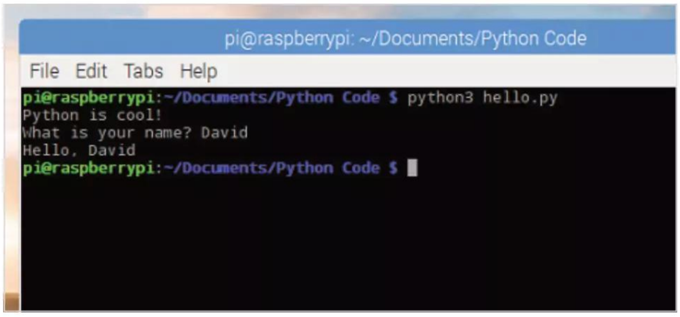
```
a=”Python”
b=”is”
c=”cool!”
print(a, b, c)
name=input("What is your name? ")
print=("Hello,", name)
```

STEP 10

Save the hello.py file and drop back to the command line. Now execute the newly saved code with:

```
python3 hello.py
```

The result will be the original Python is cool! statement, together with the added input command asking you for your name, and displaying it in the command window.


pi@raspberrypi: ~/Documents/Python Code
File Edit Tabs Help

```
pi@raspberrypi:~/Documents/Python Code $ python3 hello.py
Python is cool!
What is your name? David
Hello, David
pi@raspberrypi:~/Documents/Python Code $ 
```

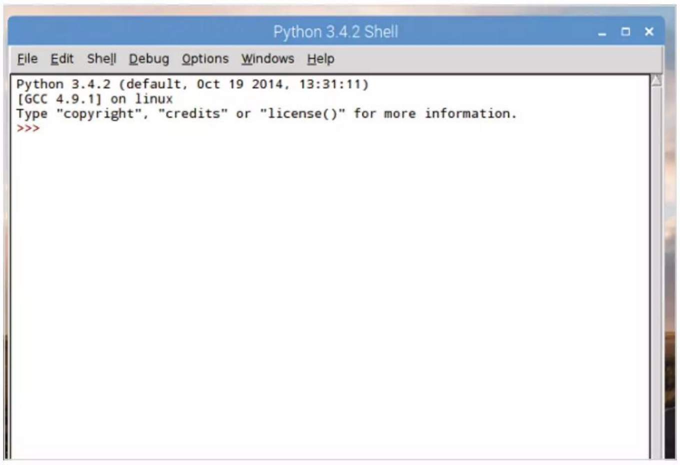
Numbers and Expressions

We've seen some basic mathematical expressions with Python, simple addition and the like. Let's expand on that now and see just how powerful Python is as a calculator. You can work within the IDLE Shell or in the Editor, whichever you like.

IT'S ALL MATHS, MAN

You can get some really impressive results with the mathematical powers of Python; as with most, if not all, programming languages, maths is the driving force behind the code.

STEP 1 Open up the GUI version of Python 3, as mentioned you can use either the Shell or the Editor. For the time being, you're going to use the Shell just to warm our maths muscle, which we believe is a small gland located at the back of the brain (or not).

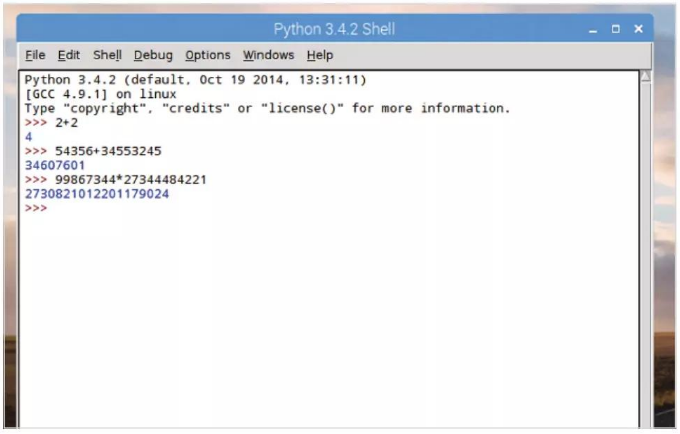

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
```

STEP 2 In the Shell enter the following:

```
2+2
54356+34553245
99867344*27344484221
```

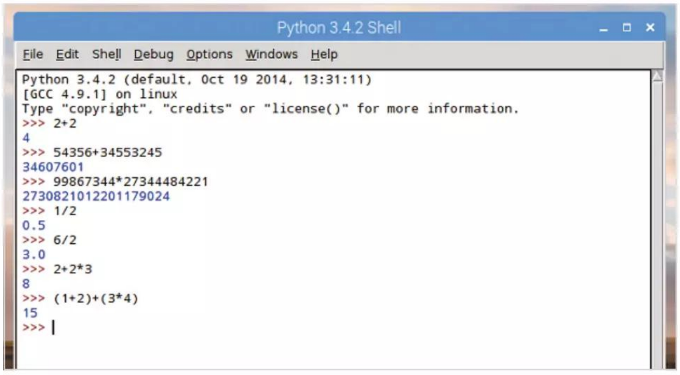
You can see that Python can handle some quite large numbers.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>> 54356+34553245
34607601
>>> 99867344*27344484221
2730821012201179024
>>>
```

STEP 3 You can use all the usual mathematical operations: divide, multiply, brackets and so on. Practise with a few, for example:

```
1/2
6/2
2+2*3
(1+2)+(3*4)
```

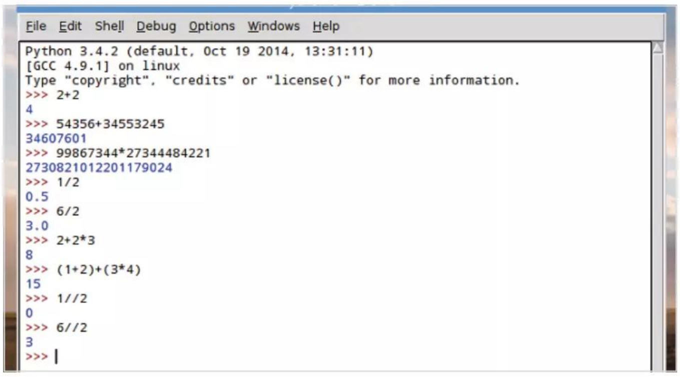

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>> 54356+34553245
34607601
>>> 99867344*27344484221
2730821012201179024
>>> 1/2
0.5
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> |
```

STEP 4 You've no doubt noticed, division produces a decimal number. In Python these are called floats, or floating point arithmetic. However, if you need an integer as opposed to a decimal answer, then you can use a double slash:

```
1//2
6//2
```

And so on.


File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>> 54356+34553245
34607601
>>> 99867344*27344484221
2730821012201179024
>>> 1/2
0.5
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>> |
```

STEP 5

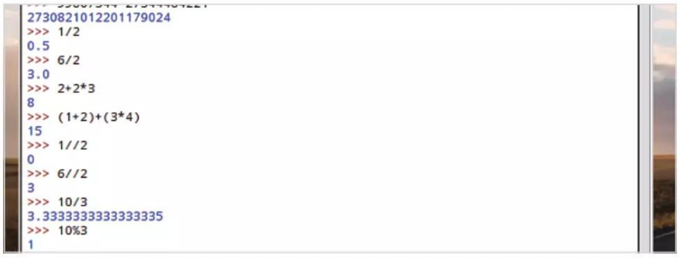
You can also use an operation to see the remainder left over from division. For example:

```
10/3
```

Will display 3.333333333, which is of course 3.3-recurring. If you now enter:

```
10%3
```

This will display 1, which is the remainder left over from dividing 10 into 3.



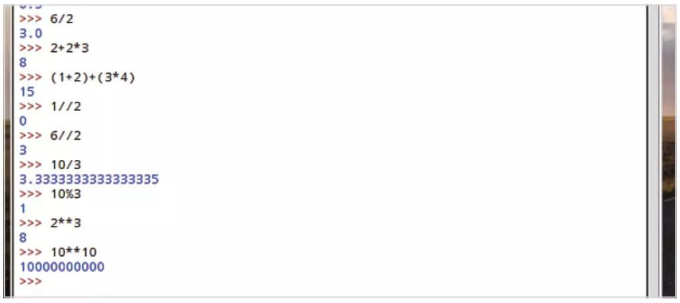
```
2730821012201179024
>>> 1/2
0.5
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>> 10/3
3.3333333333333335
>>> 10%3
1
```

STEP 6

Next up we have the power operator, or exponentiation if you want to be technical. To work out the power of something you can use a double multiplication symbol or double-star on the keyboard:

```
2**3
10**10
```

Essentially, it's 2x2x2 but we're sure you already know the basics behind maths operators. This is how you would work it out in Python.

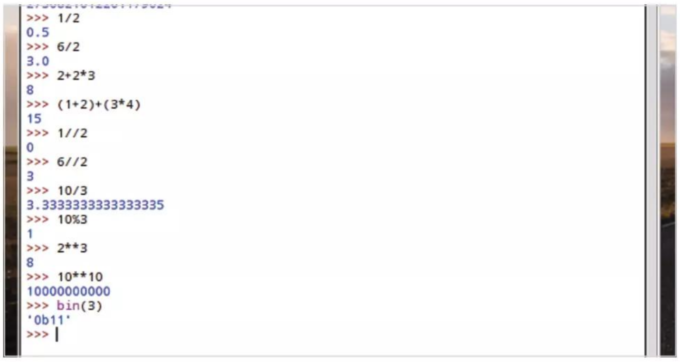


```
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>> 10/3
3.3333333333333335
>>> 10%3
1
>>> 2**3
8
>>> 10**10
10000000000
>>>
```

STEP 7

Numbers and expressions don't stop there. Python has numerous built-in functions to work out sets of numbers, absolute values, complex numbers and a host of mathematical expressions and Pythagorean tongue-twisters. For example, to convert a number to binary, use:

```
bin(3)
```



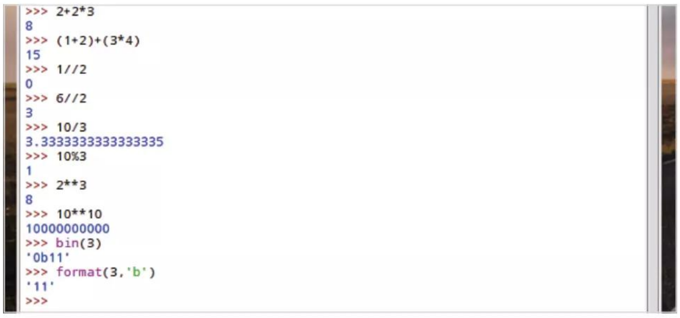
```
>>> 1/2
0.5
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>> 10/3
3.3333333333333335
>>> 10%3
1
>>> 2**3
8
>>> 10**10
10000000000
>>> bin(3)
'0b11'
>>> |
```

STEP 8

This will be displayed as '0b11', converting the integer into binary and adding the prefix 0b to the front. If you want to remove the 0b prefix, then you can use:

```
format(3, 'b')
```

The Format command converts a value, the number 3, to a formatted representation as controlled by the format specification, the 'b' part.

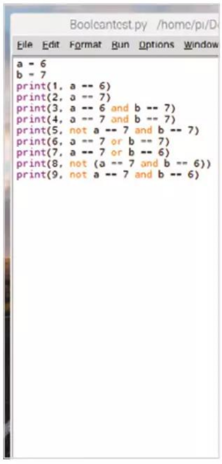


```
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>> 10/3
3.3333333333333335
>>> 10%3
1
>>> 2**3
8
>>> 10**10
10000000000
>>> bin(3)
'0b11'
>>> format(3,'b')
'11'
>>>
```

STEP 9

A Boolean Expression is a logical statement that will either be true or false. We can use these to compare data and test to see if it's equal to, less than or greater than. Try this in a New File:

```
a = 6
b = 7
print(1, a == 6)
print(2, a == 7)
print(3, a == 6 and b == 7)
print(4, a == 7 and b == 7)
print(5, not a == 7 and b == 7)
print(6, a == 7 or b == 7)
print(7, a == 7 or b == 6)
print(8, not (a == 7 and b == 6))
print(9, not a == 7 and b == 6)
```

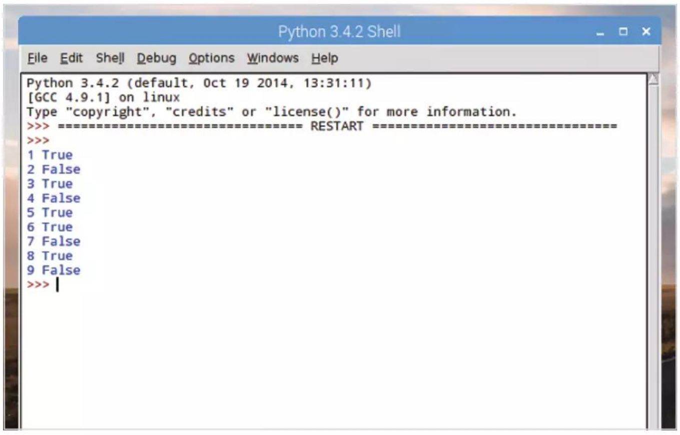


Booleantest.py /home/pi/D
File Edit Format Run Options Window

```
a = 6
b = 7
print(1, a == 6)
print(2, a == 7)
print(3, a == 6 and b == 7)
print(4, a == 7 and b == 7)
print(5, not a == 7 and b == 7)
print(6, a == 7 or b == 7)
print(7, a == 7 or b == 6)
print(8, not (a == 7 and b == 6))
print(9, not a == 7 and b == 6)
```

STEP 10

Execute the code from Step 9, and you can see a series of True or False statements, depending on the result of the two defining values: 6 and 7. It's an extension of what you've looked at, and an important part of programming.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
1 True
2 False
3 True
4 False
5 True
6 True
7 False
8 True
9 False
>>> |
```

Using Comments

When writing your code, the flow of it, what each variable does, how the overall program will operate and so on is all inside your head. Another programmer could follow the code line by line but over time, it can become difficult to read.

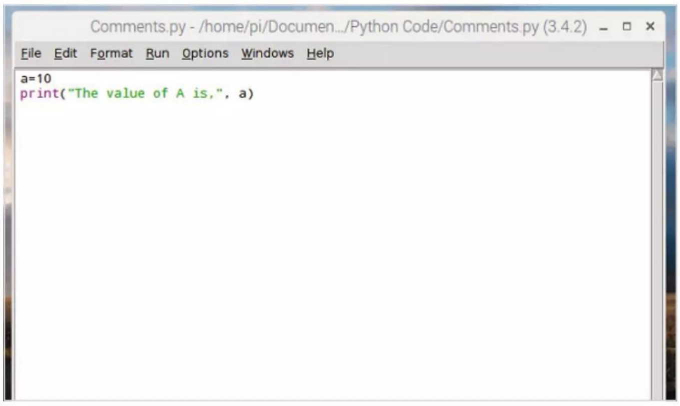
#COMMENTS!

Programmers use a method of keeping their code readable by commenting on certain sections. If a variable is used, the programmer comments on what it's supposed to do, for example. It's just good practise.

STEP 1 Start by creating a new instance of the IDLE Editor (File > New File) and create a simple variable and print command:

```
a=10
print("The value of A is,", a)
```

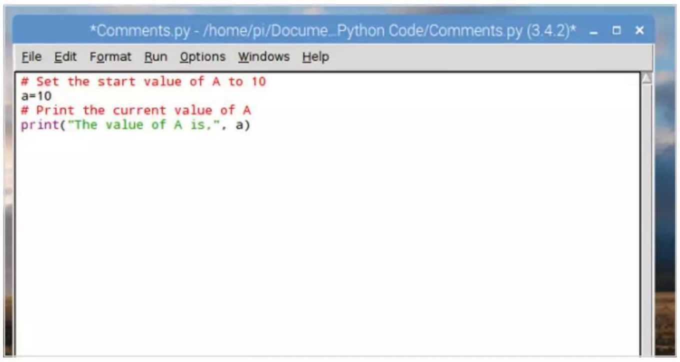
Save the file and execute the code.


Comments.py - /home/pi/Documen.../Python Code/Comments.py (3.4.2)
File
Edit
Format
Run
Options
Windows
Help

```
a=10
print("The value of A is,", a)
```

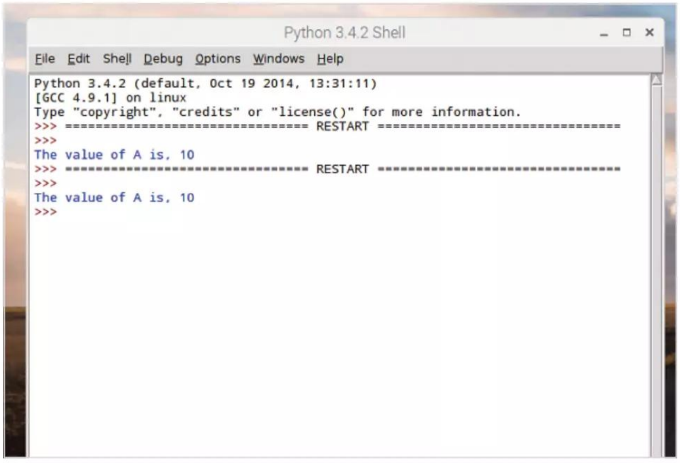
STEP 2 Running the code will return the line: The value of A is, 10 into the IDLE Shell window, which is what we expected. Now, add some of the types of comments you'd normally see within code:

```
# Set the start value of A to 10
a=10
# Print the current value of A
print("The value of A is,", a)
```


Comments.py - /home/pi/Docume...Python Code/Comments.py (3.4.2)
File
Edit
Format
Run
Options
Windows
Help

```
# Set the start value of A to 10
a=10
# Print the current value of A
print("The value of A is,", a)
```

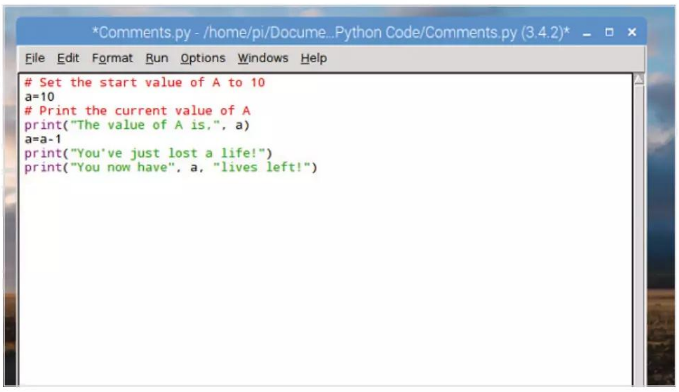
STEP 3 Resave the code and execute it. You can see that the output in the IDLE Shell is still the same as before, despite the extra lines being added. Simply put, the hash symbol (#) denotes a line of text the programmer can insert to inform them, and others, of what's going on without the user being aware.


Python 3.4.2 Shell
File
Edit
Shell
Debug
Options
Windows
Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
The value of A is, 10
>>> ================================ RESTART ================================
>>>
The value of A is, 10
>>>
```

STEP 4 Let's assume that the variable A that we've created is the number of lives in a game. Every time the player dies, the value is decreased by 1. The programmer could insert a routine along the lines of:

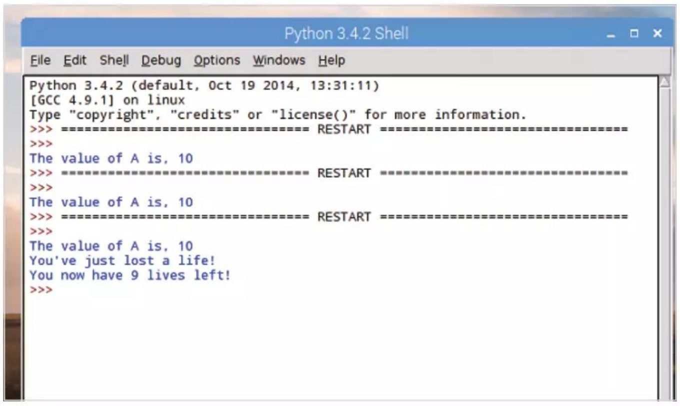
```
a=a-1
print("You've just lost a life!")
print("You now have", a, "lives left!")
```


Comments.py - /home/pi/Docume...Python Code/Comments.py (3.4.2)
File
Edit
Format
Run
Options
Windows
Help

```
# Set the start value of A to 10
a=10
# Print the current value of A
print("The value of A is,", a)
a=a-1
print("You've just lost a life!")
print("You now have", a, "lives left!")
```

STEP 5

Whilst we know that the variable A is lives, and that the player has just lost one, a casual viewer or someone checking the code may not know. Imagine for a moment that the code is twenty thousand lines long, instead of just our seven. You can see how handy comments are.

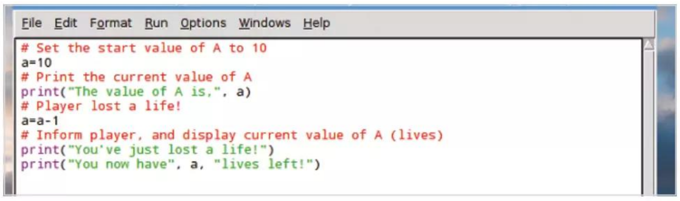

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
The value of A is, 10
>>> ================================ RESTART ================================
>>>
The value of A is, 10
>>> ================================ RESTART ================================
>>>
The value of A is, 10
You've just lost a life!
You now have 9 lives left!
>>>
```

STEP 6

Essentially, the new code together with comments could look like:

```
# Set the start value of A to 10
a=10
# Print the current value of A
print(“The value of A is,”, a)
# Player lost a life!
a=a-1
# Inform player, and display current value of A
(lives)
print(“You’ve just lost a life!”)
print(“You now have”, a, “lives left!”)
```

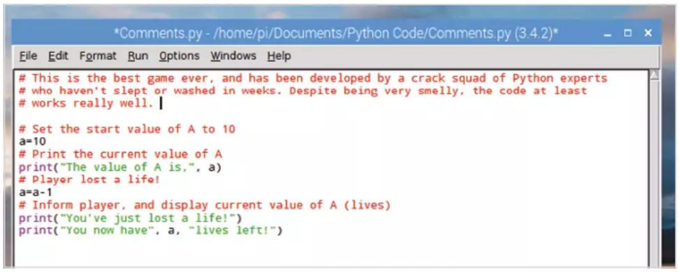

File Edit Format Run Options Windows Help

```
# Set the start value of A to 10
a=10
# Print the current value of A
print("The value of A is,", a)
# Player lost a life!
a=a-1
# Inform player, and display current value of A (lives)
print("You've just lost a life!")
print("You now have", a, "lives left!")
```

STEP 7

You can use comments in different ways. For example, Block Comments are a large section of text that details what’s going on in the code, such as telling the code reader what variables you’re planning on using:

```
# This is the best game ever, and has been
developed by a crack squad of Python experts
# who haven’t slept or washed in weeks. Despite
being very smelly, the code at least
# works really well.
```

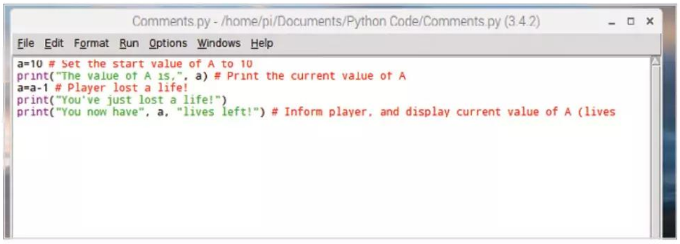

Comments.py - /home/pi/Documents/Python Code/Comments.py (3.4.2)
File Edit Format Run Options Windows Help

```
# This is the best game ever, and has been developed by a crack squad of Python experts
# who haven't slept or washed in weeks. Despite being very smelly, the code at least
# works really well.

# Set the start value of A to 10
a=10
# Print the current value of A
print("The value of A is,", a)
# Player lost a life!
a=a-1
# Inform player, and display current value of A (lives)
print("You've just lost a life!")
print("You now have", a, "lives left!")
```

STEP 8

Inline comments are comments that follow a section of code. Take our examples from above, instead of inserting the code on a separate line, we could use:

```
a=10 # Set the start value of A to 10
print(“The value of A is,”, a) # Print the current
value of A
a=a-1 # Player lost a life!
print(“You’ve just lost a life!”)
print(“You now have”, a, “lives left!”) # Inform
player, and display current value of A (lives)
```

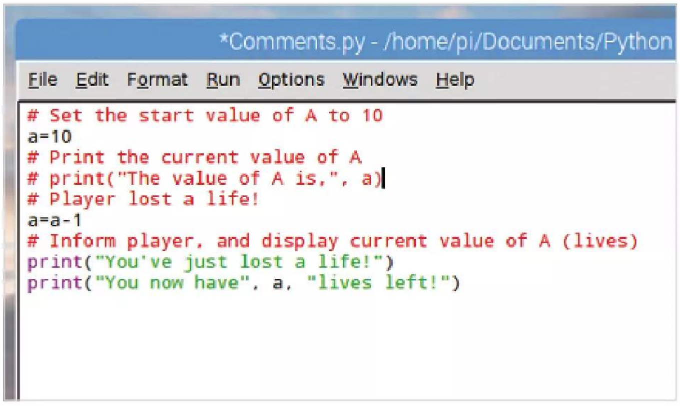

Comments.py - /home/pi/Documents/Python Code/Comments.py (3.4.2)
File Edit Format Run Options Windows Help

```
a=10 # Set the start value of A to 10
print("The value of A is,", a) # Print the current value of A
a=a-1 # Player lost a life!
print("You've just lost a life!")
print("You now have", a, "lives left!") # Inform player, and display current value of A (lives
```

STEP 9

The comment, the hash symbol, can also be used to comment out sections of code you don’t want to be executed in your program. For instance, if you wanted to remove the first print statement, you would use:

```
# print(“The value of A is,”, a)
```

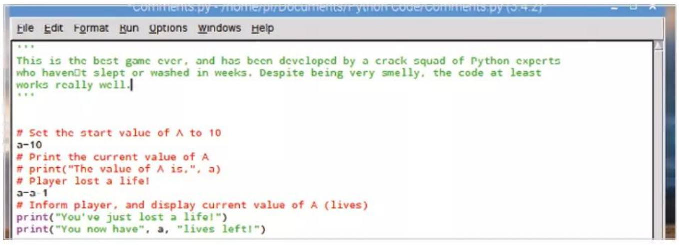

*Comments.py - /home/pi/Documents/Python
File Edit Format Run Options Windows Help

```
# Set the start value of A to 10
a=10
# Print the current value of A
# print("The value of A is,", a)
# Player lost a life!
a=a-1
# Inform player, and display current value of A (lives)
print("You've just lost a life!")
print("You now have", a, "lives left!")
```

STEP 10

You also use three single quotes to comment out a Block Comment or multi-line section of comments. Place them before and after the areas you want to comment for them to work:

```
‘’’
This is the best game ever, and has been developed
by a crack squad of Python experts who haven’t
slept or washed in weeks. Despite being very
smelly, the code at least works really well.
’’’
```


Comments.py - /home/pi/Documents/Python Code/Comments.py (3.4.2)
File Edit Format Run Options Windows Help

```
'''
This is the best game ever, and has been developed by a crack squad of Python experts
who haven□t slept or washed in weeks. Despite being very smelly, the code at least
works really well.
'''


# Set the start value of A to 10
a=10
# Print the current value of A
# print("The value of A is,", a)
# Player lost a life!
a=a-1
# Inform player, and display current value of A (lives)
print("You've just lost a life!")
print("You now have", a, "lives left!")
```

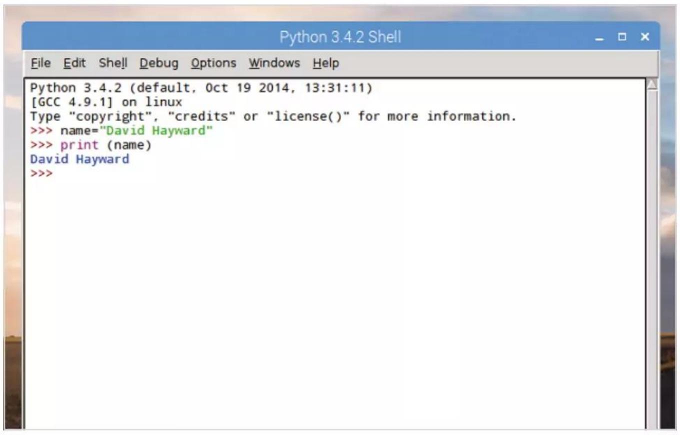
Working with Variables

We've seen some examples of variables in our Python code already but it's always worth going through the way they operate and how Python creates and assigns certain values to a variable.

VARIOUS VARIABLES

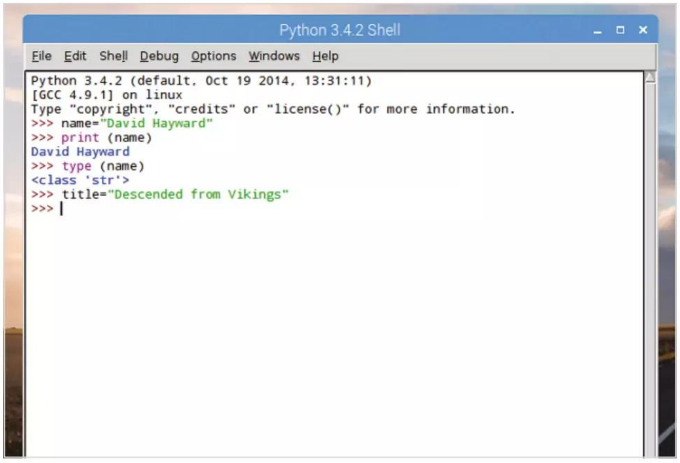
You'll be working with the Python 3 IDLE Shell in this tutorial. If you haven't already, open Python 3 or close down the previous IDLE Shell to clear up any old code.

STEP 1 In some programming languages you're required to use a dollar sign to denote a string, which is a variable made up of multiple characters, such as a name of a person. In Python this isn't necessary. For example, in the Shell enter: `name=”David Hayward”` (or use your own name, unless you're also called David Hayward).



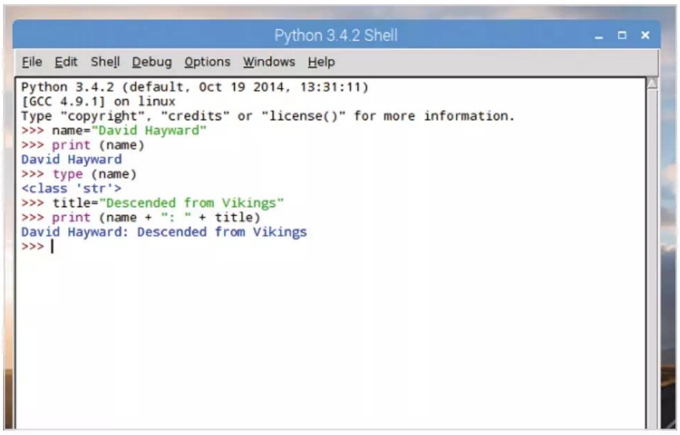
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name="David Hayward"
>>> print (name)
David Hayward
>>>
```

STEP 2 You can check the type of variable in use by issuing the **type ()** command, placing the name of the variable inside the brackets. In our example, this would be: `type (name)`. Add a new string variable: `title=”Descended from Vikings”`.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name="David Hayward"
>>> print (name)
David Hayward
>>> type (name)
<class 'str'>
>>> title="Descended from Vikings"
>>>
```

STEP 3 You've seen previously that variables can be concatenated using the plus symbol between the variable names. In our example we can use: `print (name + “: “ + title)`. The middle part between the quotations allows us to add a colon and a space, as variables are connected without spaces, so we need to add them manually.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name="David Hayward"
>>> print (name)
David Hayward
>>> type (name)
<class 'str'>
>>> title="Descended from Vikings"
>>> print (name + ": " + title)
David Hayward: Descended from Vikings
>>>
```

STEP 4 You can also combine variables within another variable. For example, to combine both name and title variables into a new variable we use:

```
character=name + “: “ +
title
```

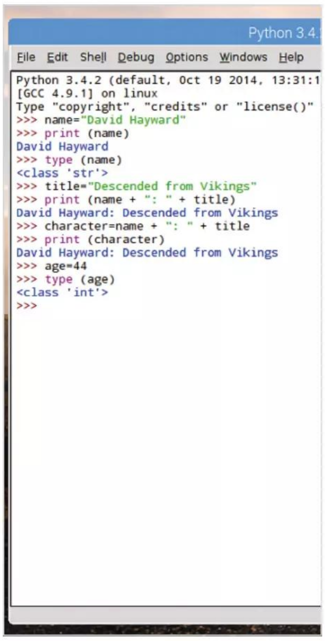
Then output the content of the new variable as:

```
print (character)
```

Numbers are stored as different variables:

```
age=44
Type (age)
```

Which, as we know, are integers.

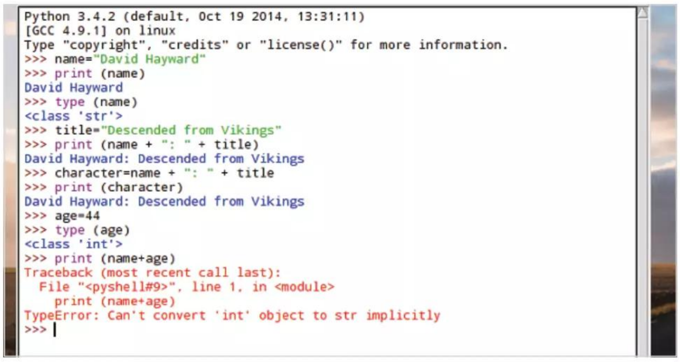


```
Python 3.4.2 (default, Oct 19 2014, 13:31:1
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()"
>>> name="David Hayward"
>>> print (name)
David Hayward
>>> type (name)
<class 'str'>
>>> title="Descended from Vikings"
>>> print (name + ": " + title)
David Hayward: Descended from Vikings
>>> character=name + ": " + title
>>> print (character)
David Hayward: Descended from Vikings
>>> age=44
>>> type (age)
<class 'int'>
>>>
```

STEP 5

However, you can't combine both strings and integer type variables in the same command, as you would a set of similar variables. You need to either turn one into the other or vice versa. When you do try to combine both, you get an error message:

`print (name + age)`



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name="David Hayward"
>>> print (name)
David Hayward
>>> type (name)
<class 'str'>
>>> title="Descended from Vikings"
>>> print (name + ": " + title)
David Hayward: Descended from Vikings
>>> character=name + ": " + title
>>> print (character)
David Hayward: Descended from Vikings
>>> age=44
>>> type (age)
<class 'int'>
>>> print (name+age)
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    print (name+age)
TypeError: Can't convert 'int' object to str implicitly
>>> 
```

STEP 6

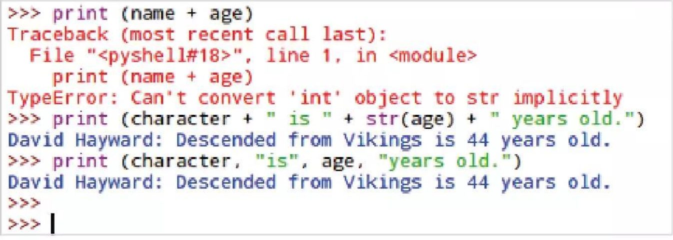
This is a process known as TypeCasting. The Python code is:

`print (character + “ is “ + str(age) + “ years old.”)`

or you can use:

`print (character, “is“, age, “years old.”)`

Notice again that in the last example, you don't need the spaces between the words in quotes as the commas treat each argument to print separately.



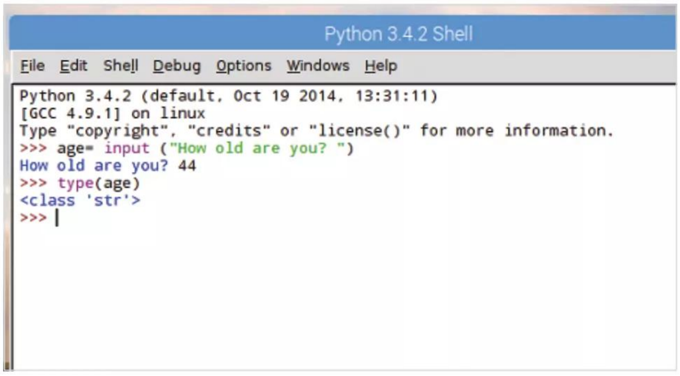
```
>>> print (name + age)
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    print (name + age)
TypeError: Can't convert 'int' object to str implicitly
>>> print (character + " is " + str(age) + " years old.")
David Hayward: Descended from Vikings is 44 years old.
>>> print (character, "is", age, "years old.")
David Hayward: Descended from Vikings is 44 years old.
>>>
>>> 
```

STEP 7

Another example of TypeCasting is when you ask for input from the user, such as a name. for example, enter:

`age= input (“How old are you? “)`

All data stored from the Input command is stored as a string variable.



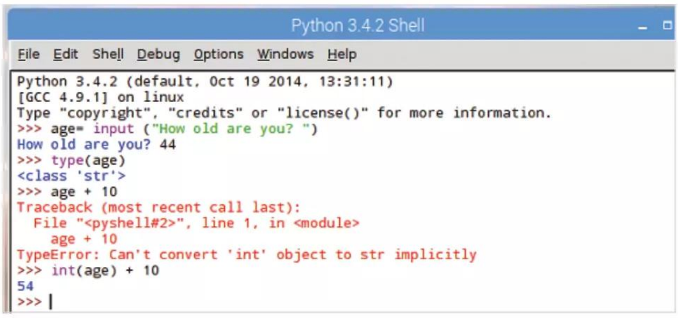
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> age= input ("How old are you? ")
How old are you? 44
>>> type(age)
<class 'str'>
>>> 
```

STEP 8

This presents a bit of a problem when you want to work with a number that's been inputted by the user, as age + 10 won't work due to being a string variable and an integer. Instead, you need to enter:

`int(age) + 10`

This will TypeCast the age string into an integer that can be worked with.



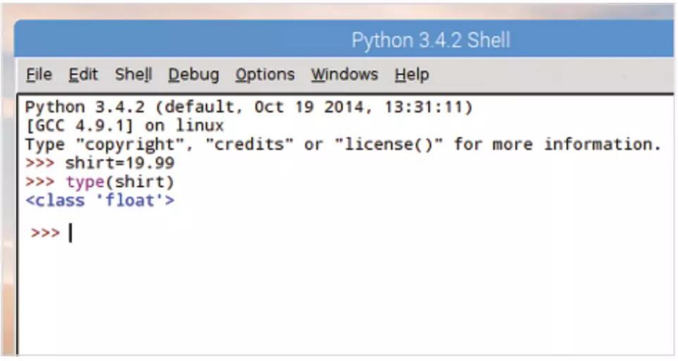
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> age= input ("How old are you? ")
How old are you? 44
>>> type(age)
<class 'str'>
>>> age + 10
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    age + 10
TypeError: Can't convert 'int' object to str implicitly
>>> int(age) + 10
54
>>> 
```

STEP 9

The use of TypeCasting is also important when dealing with floating point arithmetic; remember: numbers that have a decimal point in them. For example, enter:

`shirt=19.99`

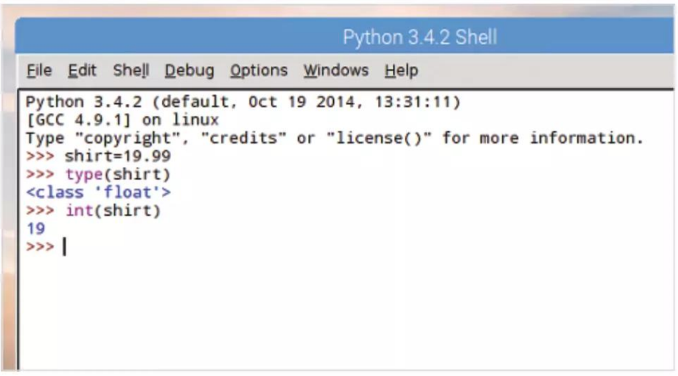
Now enter `type(shirt)` and you'll see that Python has allocated the number as a 'float', because the value contains a decimal point.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> shirt=19.99
>>> type(shirt)
<class 'float'>

 >>> 
```

STEP 10

When combining integers and floats Python usually converts the integer to a float, but should the reverse ever be applied it's worth remembering that Python doesn't return the exact value. When converting a float to an integer, Python will always round down to the nearest integer, called truncating; in our case instead of 19.99 it becomes 19.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> shirt=19.99
>>> type(shirt)
<class 'float'>
>>> int(shirt)
19
>>> 
```

User Input

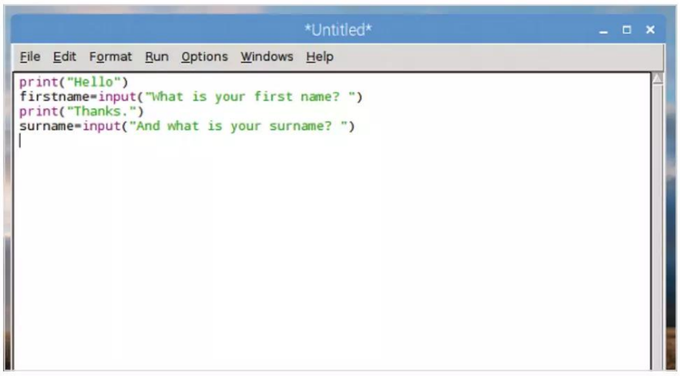
We've seen some basic user interaction with the code from a few of the examples earlier, so now would be a good time to focus solely on how you would get information from the user then store and present it.

USER FRIENDLY

The type of input you want from the user will depend greatly on the type of program you're coding. For example, a game may ask for a character's name, whereas a database can ask for personal details.

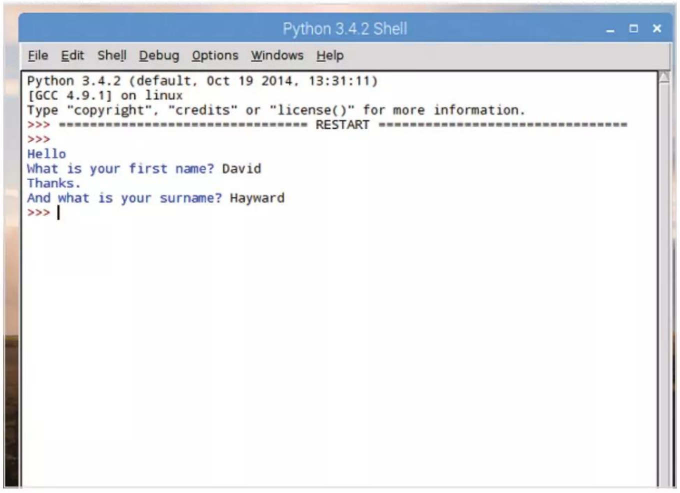
STEP 1 If it's not already, open the Python 3 IDLE Shell, and start a New File in the Editor. Let's begin with something really simple, enter:

```
print("Hello")
firstname=input("What is your first name? ")
print("Thanks.")
surname=input("And what is your surname? ")
```


Untitled
File Edit Format Run Options Windows Help

```
print("Hello")
firstname=input("What is your first name? ")
print("Thanks.")
surname=input("And what is your surname? ")
```

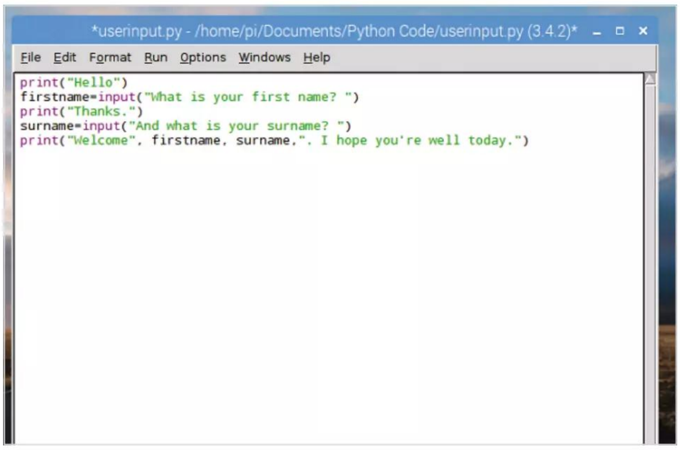
STEP 2 Save and execute the code, and as you already no doubt suspected, in the IDLE Shell the program will ask for your first name, storing it as the variable firstname, followed by your surname; also stored in its own variable (surname).


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Hello
What is your first name? David
Thanks.
And what is your surname? Hayward
>>> 
```

STEP 3 Now that we have the user's name stored in a couple of variables we can call them up whenever we want:

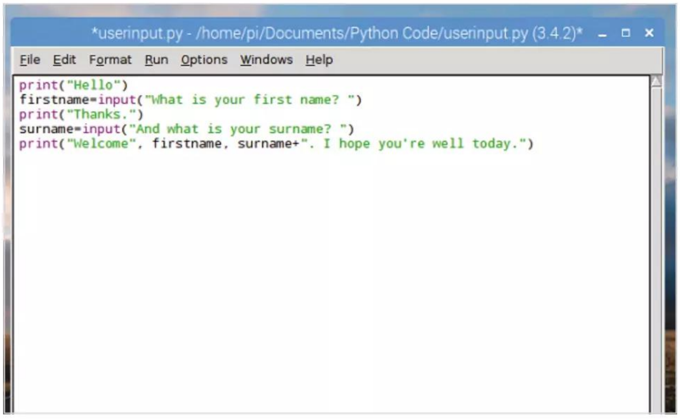
```
print("Welcome", firstname, surname, ". I hope
you're well today.")
```


userinput.py - /home/pi/Documents/Python Code/userinput.py (3.4.2)
File Edit Format Run Options Windows Help

```
print("Hello")
firstname=input("What is your first name? ")
print("Thanks.")
surname=input("And what is your surname? ")
print("Welcome", firstname, surname,". I hope you're well today.")
```

STEP 4 Run the code and you can see a slight issue, the full stop after the surname follows a blank space. To eliminate that we can add a plus sign instead of the comma in the code:

```
print("Welcome", firstname, surname+". I hope
you're well today.")
```

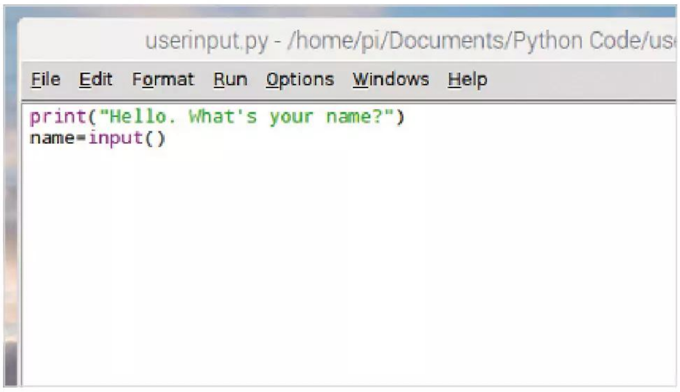

userinput.py - /home/pi/Documents/Python Code/userinput.py (3.4.2)
File Edit Format Run Options Windows Help

```
print("Hello")
firstname=input("What is your first name? ")
print("Thanks.")
surname=input("And what is your surname? ")
print("Welcome", firstname, surname+". I hope you're well today.")
```

STEP 5

You don't always have to include quoted text within the input command. For example, you can ask the user their name, and have the input in the line below:

```
print("Hello. What's your name?")
name=input()
```

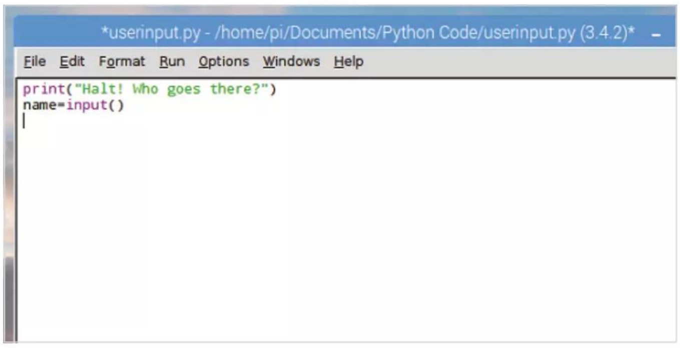

userinput.py - /home/pi/Documents/Python Code/us
File Edit Format Run Options Windows Help

```
print("Hello. What's your name?")
name=input()
```

STEP 6

The code from the previous step is often regarded as being a little neater than having a lengthy amount of text in the input command, but it's not a rule that's set in stone, so do as you like in these situations. Expanding on the code, try this:

```
print("Halt! Who goes there?")
name=input()
```

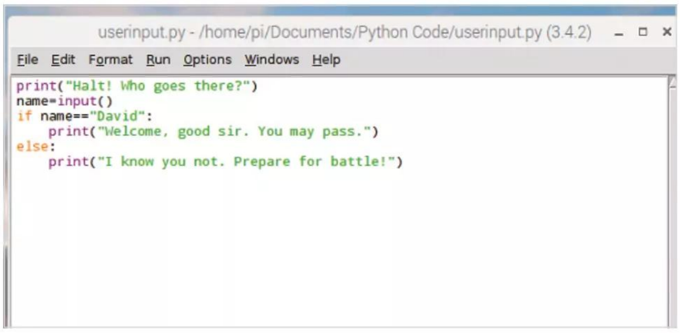

userinput.py - /home/pi/Documents/Python Code/userinput.py (3.4.2)
File Edit Format Run Options Windows Help

```
print("Halt! Who goes there?")
name=input()
```

STEP 7

It's a good start to a text adventure game, perhaps? Now you can expand on it and use the raw input from the user to flesh out the game a little:

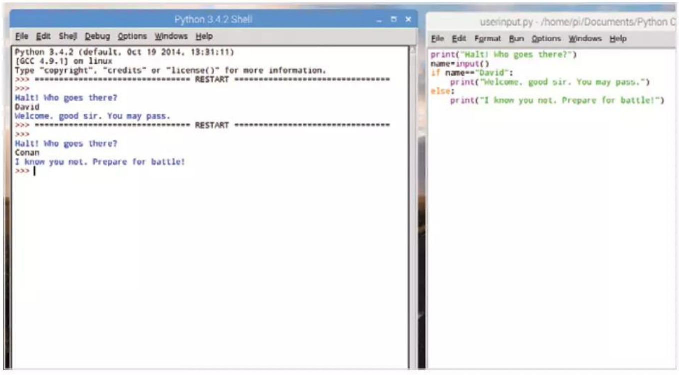
```
if name=="David":
  print("Welcome, good sir. You may pass.")
else:
 print("I know you not. Prepare for battle!")
```


userinput.py - /home/pi/Documents/Python Code/userinput.py (3.4.2)
File Edit Format Run Options Windows Help

```
print("Halt! Who goes there?")
name=input()
if name=="David":
    print("Welcome, good sir. You may pass.")
else:
    print("I know you not. Prepare for battle!")
```

STEP 8

What you've created here is a condition, which we will cover soon. In short, we're using the input from the user and measuring it against a condition. So, if the user enters David as their name, the guard will allow them to pass unhindered. Else, if they enter a name other than David, the guard challenges them to a fight.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
userinput.py - /home/pi/Documents/Python C
File Edit Format Run Options Windows Help

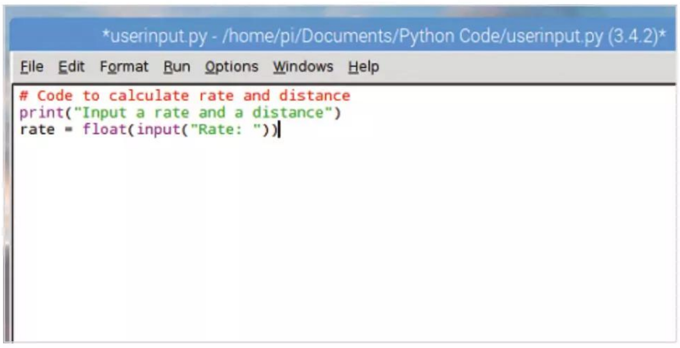
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Halt! Who goes there?
David
Welcome, good sir. You may pass.
>>> ================================ RESTART ================================
>>>
Halt! Who goes there?
Conan
I know you not. Prepare for battle!
>>>
```

```
print("Halt! Who goes there?")
name=input()
if name=="David":
    print("Welcome, good sir. You may pass.")
else:
    print("I know you not. Prepare for battle!")
```

STEP 9

Just as you learned previously, any input from a user is automatically a string, so you need to apply a TypeCast in order to turn it into something else. This creates some interesting additions to the input command. For example:

```
# Code to calculate rate and distance
print("Input a rate and a distance")
rate = float(input("Rate: "))
```


userinput.py - /home/pi/Documents/Python Code/userinput.py (3.4.2)
File Edit Format Run Options Windows Help

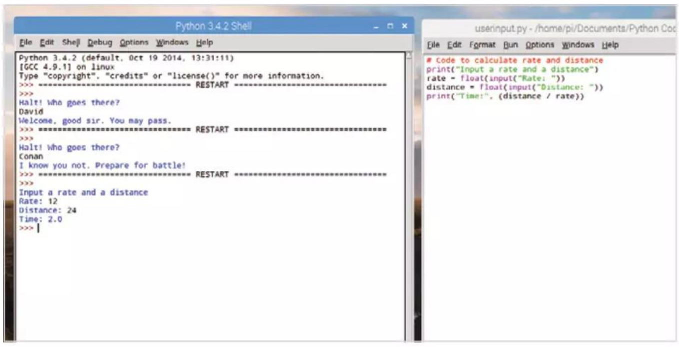
```
# Code to calculate rate and distance
print("Input a rate and a distance")
rate = float(input("Rate: "))
```

STEP 10

To finalise the rate and distance code, we can add:

```
distance = float(input("Distance: "))
print("Time:", (distance / rate))
```

Save and execute the code and enter some numbers. Using the float(input element, we've told Python that anything entered is a floating point number rather than a string.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
userinput.py - /home/pi/Documents/Python Coc
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Halt! Who goes there?
David
Welcome, good sir. You may pass.
>>> ================================ RESTART ================================
>>>
Halt! Who goes there?
Conan
I know you not. Prepare for battle!
>>> ================================ RESTART ================================
>>>
Input a rate and a distance
Rate: 12
Distance: 24
Time: 2.0
>>>
```

```
# Code to calculate rate and distance
print("Input a rate and a distance")
rate = float(input("Rate: "))
distance = float(input("Distance: "))
print("Time:", (distance / rate))
```

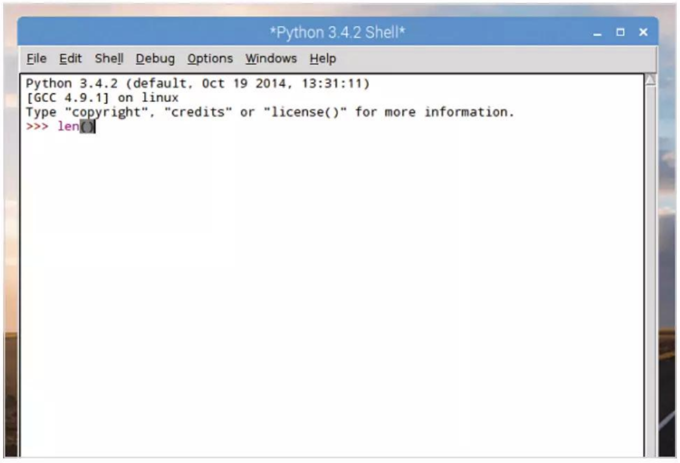
Creating Functions

Now that you've mastered the use of variables and user input, the next step is to tackle functions. You've already used a few functions, such as the print command but Python enables you to define your own functions.

FUNKY FUNCTIONS

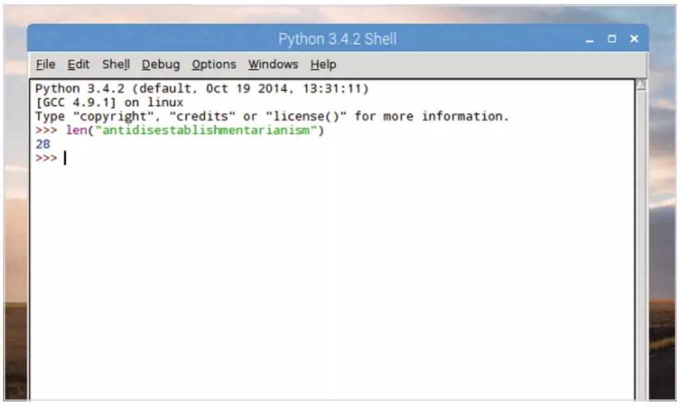
A function is a command that you enter into Python to do something. It's a little piece of self-contained code that takes data, works on it and then returns the result.

STEP 1 It's not just data that a function works on. They can do all manner of useful things in Python, such as sort data, change items from one format to another and check the length or type of items. Basically, a function is a short word that's followed by brackets. For example, **len()**, **list()** or **type()**.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> len()
```

STEP 2 A function takes data, usually a variable, works on it depending on what the function is programmed to do and returns the end value. The data being worked on goes inside the brackets, so if you wanted to know how many letters are in the word antidisestablishmentarianism, then you'd enter: `len("antidisestablishmentarianism")` and the number 28 would return.

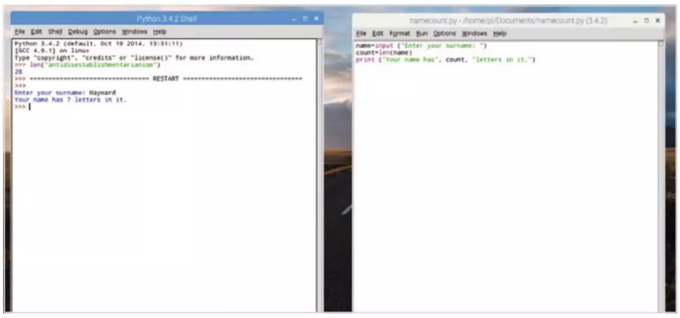

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> len("antidisestablishmentarianism")
28
>>> |
```

STEP 3 You can pass variables through functions in much the same manner. Let's assume you want the number of letters in a person's surname, you could use the following code (enter the text editor for this example):

```
name=input (“Enter your surname: “)
count=len(name)
print (“Your surname has“, count, “letters in
it.”)
```

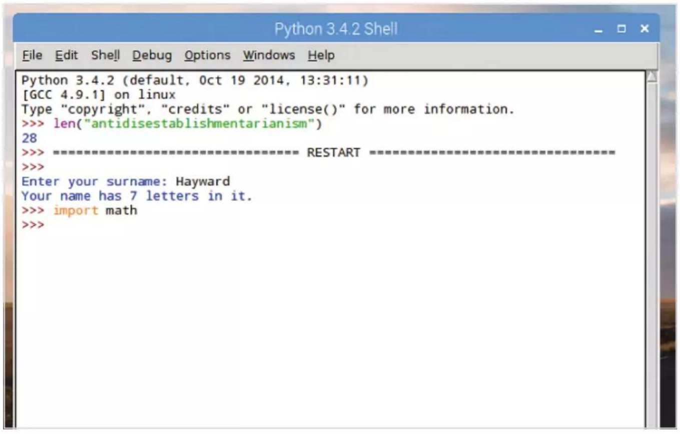
Press F5 and save the code to execute it.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
namecount.py - /home/pi/Documents/namecount.py (3.4.2)
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> len("antidisestablishmentarianism")
28
>>> ================================ RESTART ================================
>>>
Enter your surname: Hayward
Your name has 7 letters in it.
>>> |
```

```
name=input ("Enter your surname: ")
count=len(name)
print ("Your name has", count, "letters in it.")
```

STEP 4 Python has tens of functions built into it, far too many to get into in the limited space available here. However, to view the list of built-in functions available to Python 3, navigate to **www.docs.python.org/3/library/functions.html**. These are the predefined functions, but since users have created many more, they're not the only ones available.

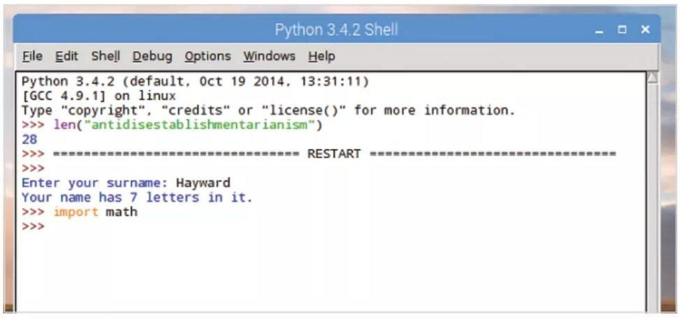

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> len("antidisestablishmentarianism")
28
>>> ================================ RESTART ================================
>>>
Enter your surname: Hayward
Your name has 7 letters in it.
>>> import math
>>>
```

STEP 5 Additional functions can be added to Python through modules. Python has a vast range of modules available that can cover numerous programming duties. They add functions and can be imported as and when required. For example, to use advanced mathematics functions enter:

```
import math
```

Once entered, you have access to all the math module functions.

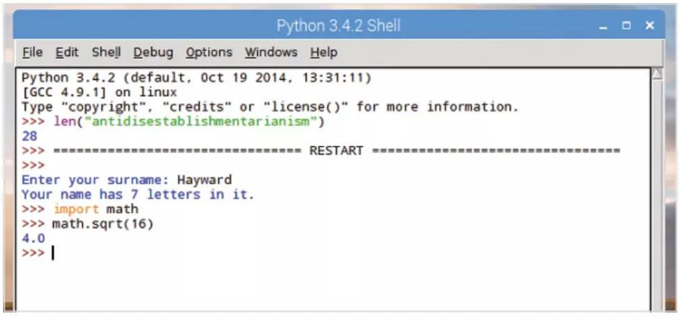

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> len("antidisestablishmentarianism")
28
>>> ================================ RESTART ================================
>>>
Enter your surname: Hayward
Your name has 7 letters in it.
>>> import math
>>>
```

STEP 6 To use a function from a module enter the name of the module followed by a full stop, then the name of the function. For instance, using the math module, since you've just imported it into Python, you can utilise the square root function. To do so, enter:

```
math.sqrt(16)
```

You can see that the code is presented as module.function(data).


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> len("antidisestablishmentarianism")
28
>>> ================================ RESTART ================================
>>>
Enter your surname: Hayward
Your name has 7 letters in it.
>>> import math
>>> math.sqrt(16)
4.0
>>> |
```

FORGING FUNCTIONS

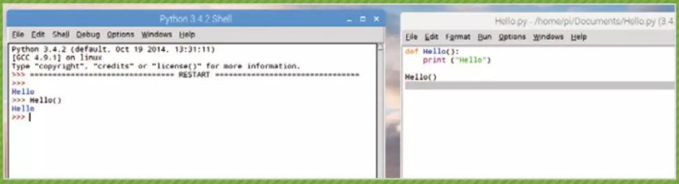
There are many different functions you can import created by other Python programmers and you will undoubtedly come across some excellent examples in the future; you can also create your own with the def command.

STEP 1 Choose File > New File to enter the editor, let's create a function called Hello, that greets a user. Enter:

```
def Hello():
   print (“Hello”)
```

```
Hello()
```

Press F5 to save and run the script. You can see Hello in the Shell, type in Hello() and it returns the new function.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Hello.py - /home/pi/Documents/Hello.py (3.4
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Hello
>>> Hello()
Hello
>>> |
```

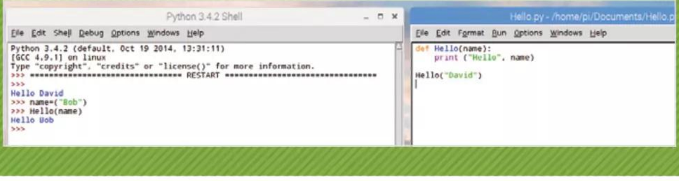
```
def Hello():
    print ("Hello")

Hello()
```

STEP 2 Let's now expand the function to accept a variable, the user's name for example. Edit your script to read:

```
def Hello(name):
   print (“Hello”, name)
```

```
Hello(“David”)
```

This will now accept the variable name, otherwise it prints Hello David. In the Shell, enter: name=(“Bob”), then: Hello(name). Your function can now pass variables through it.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Hello.py - /home/pi/Documents/Hello.p
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Hello David
>>> name=("Bob")
>>> Hello(name)
Hello Bob
>>>
```

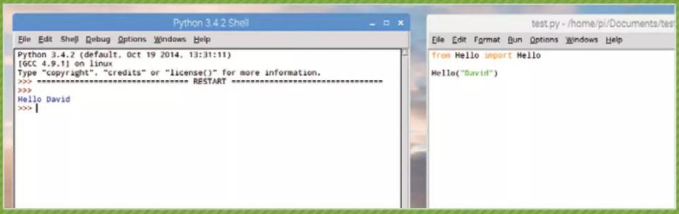
```
def Hello(name):
    print ("Hello", name)

Hello("David")
|
```

STEP 3 To modify it further, delete the Hello(“David”) line, the last line in the script and press Ctrl+S to save the new script. Close the Editor and create a new file (File > New File). Enter the following:

```
from Hello import Hello
```

```
Hello(“David”)
```

Press F5 to save and execute the code.

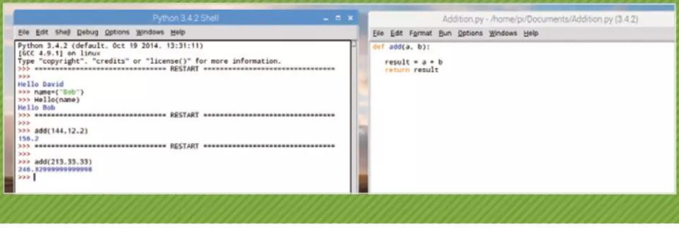

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
test.py - /home/pi/Documents/tes
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Hello David
>>> |
```

```
from Hello import Hello

Hello("David")
```

STEP 4 What you've just done is import the Hello function from the saved Hello.py program and then used it to say hello to David. This is how modules and functions work: you import the module then use the function. Try this one, and modify it for extra credit:

```
def add(a, b):

 result = a + b
 return result
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Addition.py - /home/pi/Documents/Addition.py (3.4.2)
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Hello David
>>> name=("Bob")
>>> Hello(name)
Hello Bob
>>> ================================ RESTART ================================
>>>
>>> add(144,12.2)
156.2
>>> ================================ RESTART ================================
>>>
>>> add(213,33.33)
246.32999999999998
>>> |
```

```
def add(a, b):

   result = a + b
   return result
```

Conditions and Loops

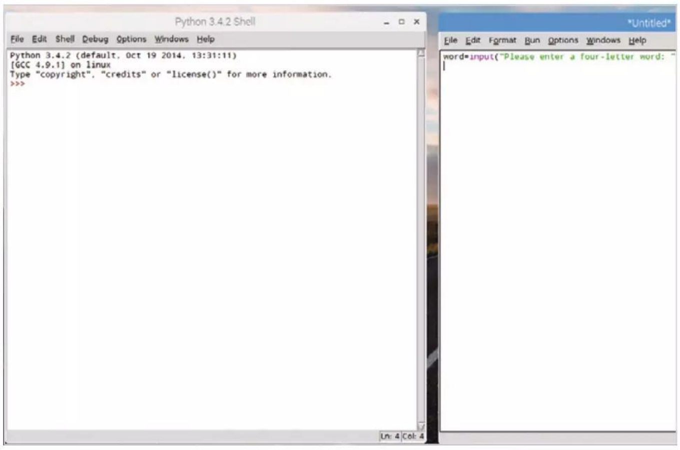
Conditions and loops are what makes a program interesting; they can be simple or rather complex. How you use them depends greatly on what the program is trying to achieve; they could be the number of lives left in a game or just displaying a countdown.

TRUE CONDITIONS

Keeping conditions simple to begin with makes learning to program a more enjoyable experience. Let's start then by checking if something is TRUE, then doing something else if it isn't.

STEP 1 Let's create a new Python program that will ask the user to input a word, then check it to see if it's a four-letter word or not. Start with File > New File, and begin with the input variable:

```
word=input("Please enter a four-letter word: ")
```

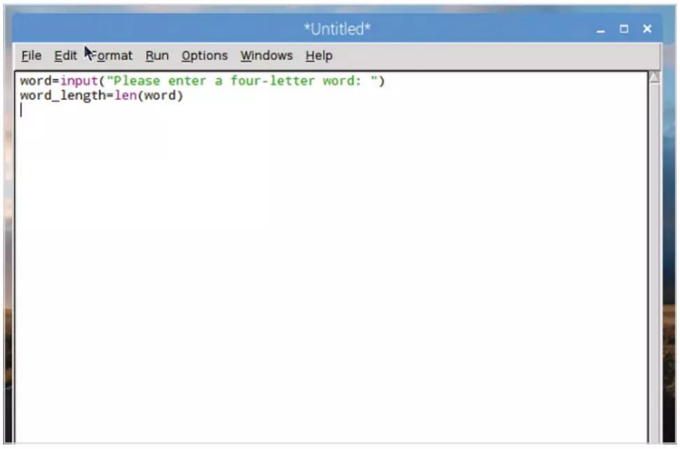

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Untitled
File Edit Format Run Options Windows Help
Ln: 4 Col: 4

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
```

```
word=input("Please enter a four-letter word: "
```

STEP 2 Now we can create a new variable, then use the len function and pass the word variable through it to get the total number of letters the user has just entered:

```
word=input("Please enter a four-letter word: ")
word_length=len(word)
```

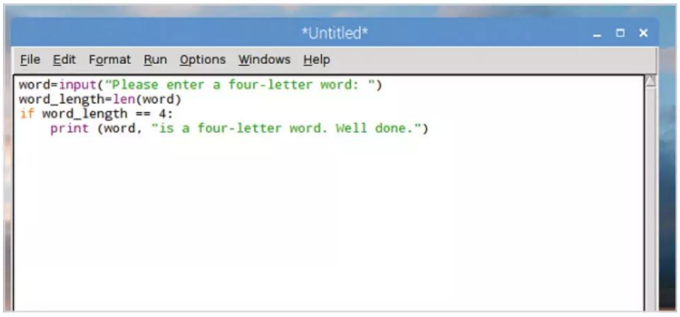

Untitled
File Edit Format Run Options Windows Help

```
word=input("Please enter a four-letter word: ")
word_length=len(word)
```

STEP 3 Now you can use an if statement to check if the word_length variable is equal to four and print a friendly conformation if it applies to the rule:

```
word=input("Please enter a four-letter word: ")
word_length=len(word)
if word_length == 4:
   print (word, "is a four-letter word. Well done.")
```

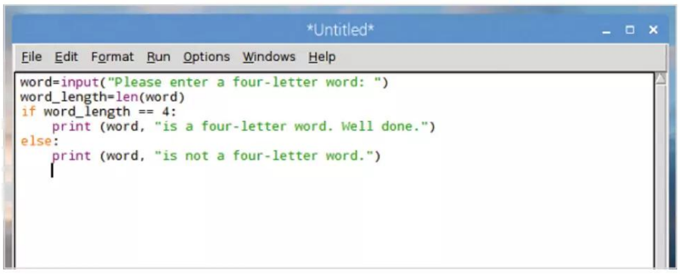
The double equal sign (==) means check if something is equal to something else.


Untitled
File Edit Format Run Options Windows Help

```
word=input("Please enter a four-letter word: ")
word_length=len(word)
if word_length == 4:
    print (word, "is a four-letter word. Well done.")
```

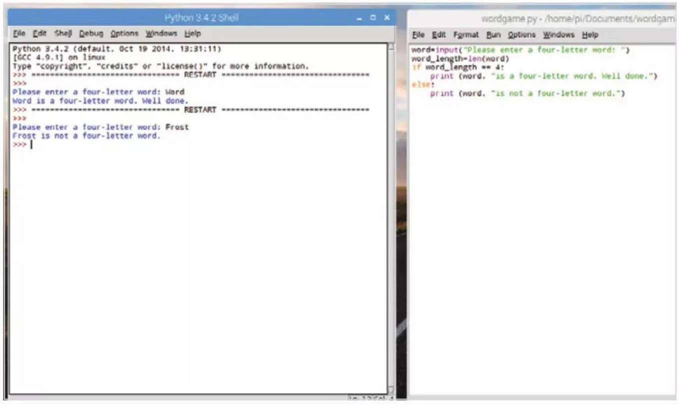
STEP 4 The colon at the end of IF tells Python that if this statement is true do everything after the colon that's indented. Next, move the cursor back to the beginning of the Editor:

```
word=input("Please enter a four-letter word: ")
word_length=len(word)
if word_length == 4:
   print (word, "is a four-letter word. Well
done.")
else:
   print (word, "is not a four-letter word.")
```


Untitled
File Edit Format Run Options Windows Help

```
word=input("Please enter a four-letter word: ")
word_length=len(word)
if word_length == 4:
    print (word, "is a four-letter word. Well done.")
else:
    print (word, "is not a four-letter word.")
```

STEP 5 Press F5 and save the code to execute it. Enter a four-letter word in the Shell to begin with, you should have the returned message that it's the word is four letters. Now press F5 again and rerun the program but this time enter a five-letter word. The Shell will display that it's not a four-letter word.

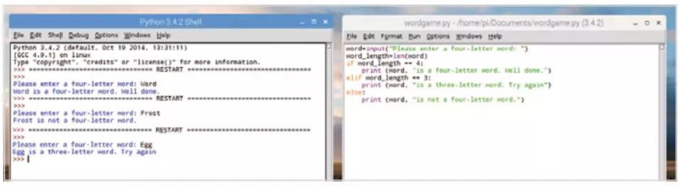

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
wordgame.py - /home/pi/Documents/wordgam
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Please enter a four-letter word: Word
Word is a four-letter word. Well done.
>>> ================================ RESTART ================================
>>>
Please enter a four-letter word: Frost
Frost is not a four-letter word.
>>> |
```

```
word=input("Please enter a four-letter word: ")
word_length=len(word)
if word_length == 4:
    print (word, "is a four-letter word. Well done.")
else:
    print (word, "is not a four-letter word.")
```

STEP 6 Now expand the code to include another conditions. Eventually, it could become quite complex. We've added a condition for three-letter words:

```
word=input("Please enter a four-letter word: ")
word_length=len(word)
if word_length == 4:
   print (word, "is a four-letter word. Well 
done.")
elif word_length == 3:
   print (word, "is a three-letter word. Try again.")
else:
   print (word, "is not a four-letter word.")
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
wordgame.py - /home/pi/Documents/wordgame.py (3.4.2)
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Please enter a four-letter word: Word
Word is a four-letter word. Well done.
>>> ================================ RESTART ================================
>>>
Please enter a four-letter word: Frost
Frost is not a four-letter word.
>>> ================================ RESTART ================================
>>>
Please enter a four-letter word: Egg
Egg is a three-letter word. Try again
>>> |
```

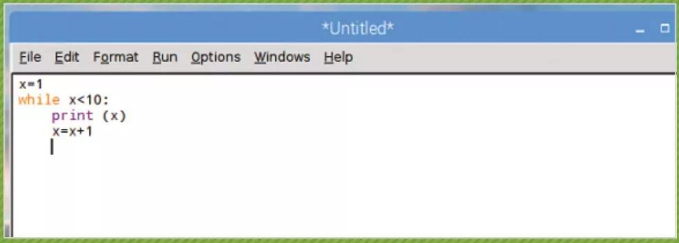
```
word=input("Please enter a four-letter word: ")
word_length=len(word)
if word_length == 4:
    print (word, "is a four-letter word. Well done.")
elif word_length == 3:
    print (word, "is a three-letter word. Try again")
else:
    print (word, "is not a four-letter word.")
```

LOOPS

A loop looks quite similar to a condition but they are somewhat different in their operation. A loop will run through the same block of code a number of times, usually with the support of a condition.

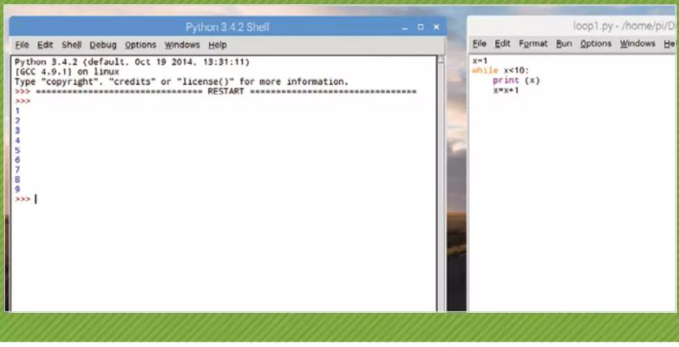
STEP 1 Let's start with a simple While statement. Like IF, this will check to see if something is TRUE, then run the indented code:

```
x = 1
while x < 10:
  print (x)
  x = x + 1
```


Untitled
File Edit Format Run Options Windows Help

```
x=1
while x<10:
    print (x)
    x=x+1
    |
```

STEP 2 The difference between if and while is when while gets to the end of the indented code, it goes back and checks the statement is still true. In our example x is less than 10. With each loop it prints the current value of x, then adds one to that value. When x does eventually equal 10 it stops.

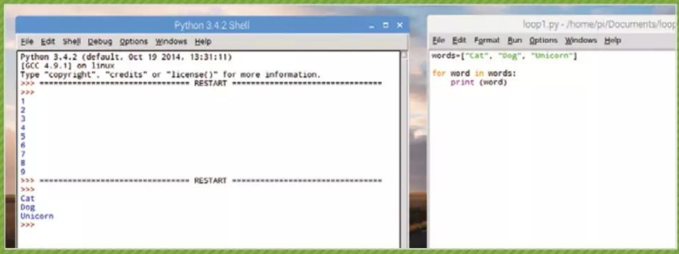

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
loop1.py - /home/pi/D
File Edit Format Run Options Windows He

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
1
2
3
4
5
6
7
8
9
>>> |
```

```
x=1
while x<10:
    print (x)
    x=x+1
```

STEP 3 The For loop is another example. For is used to loop over a range of data, usually a list stored as variables inside square brackets. For example:

```
words=["Cat", "Dog", "Unicorn"]

for word in words:
  print (word)
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
loop1.py - /home/pi/Documents/loop
File Edit Format Run Options Windows Help

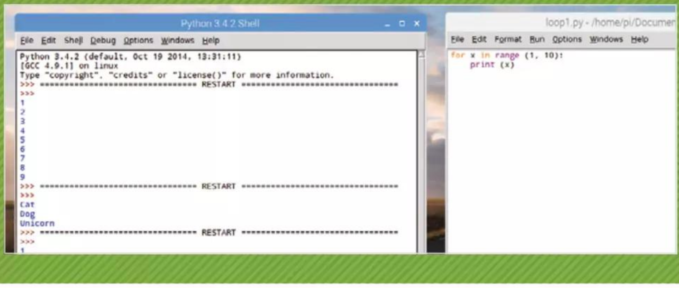
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
1
2
3
4
5
6
7
8
9
>>> ================================ RESTART ================================
>>>
Cat
Dog
Unicorn
>>>
```

```
words=["Cat", "Dog", "Unicorn"]

for word in words:
    print (word)
```

STEP 4 The For loop can also be used in the countdown example by using the range function:

```
for x in range (1, 10):
 print (x)
```

The x=x+1 part isn't needed here because the range function creates a list between the first and last numbers used.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
loop1.py - /home/pi/Documer
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
1
2
3
4
5
6
7
8
9
>>> ================================ RESTART ================================
>>>
Cat
Dog
Unicorn
>>> ================================ RESTART ================================
>>>
1
```

```
for x in range (1, 10):
    print (x)
```

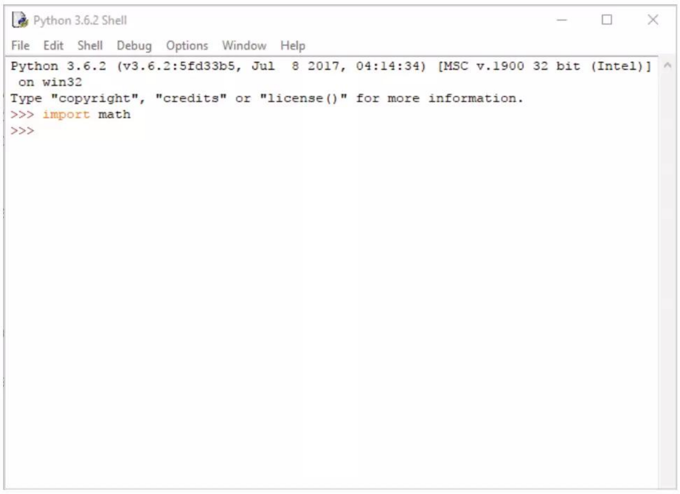
Python Modules

We've mentioned modules previously, (the Math module) but as modules are such a large part of getting the most from Python, it's worth dedicating a little more time to them. In this instance we're using the Windows version of Python 3.

MASTERING MODULES

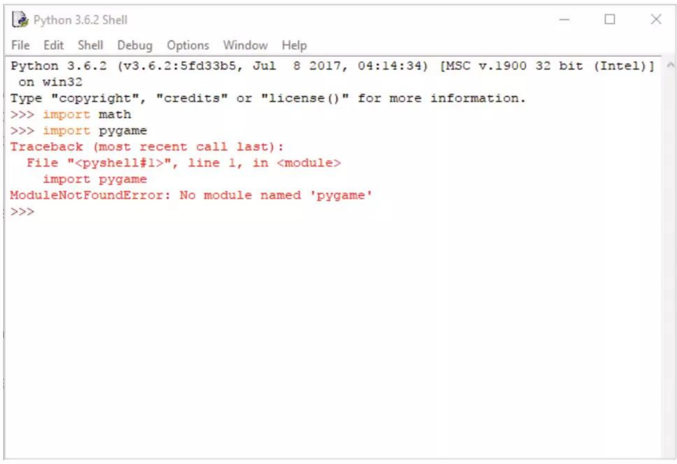
Think of modules as an extension that's imported into your Python code to enhance and extend its capabilities. There are countless modules available and as we've seen, you can even make your own.

STEP 1 Although good, the built-in functions within Python are limited. The use of modules, however, allows us to make more sophisticated programs. As you are aware, modules are Python scripts that are imported, such as import math.



```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> import math
>>>
```

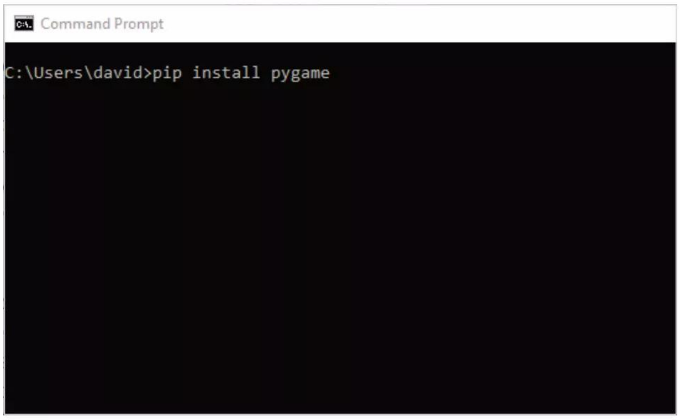
STEP 2 Some modules, especially on the Raspberry Pi, are included by default, the math module being a prime example. Sadly, other modules aren't always available. A good example on non-Pi platforms is the pygame module, which contains many functions to help create games. Try: **import pygame**.



```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> import math
>>> import pygame
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    import pygame
ModuleNotFoundError: No module named 'pygame'
>>>
```

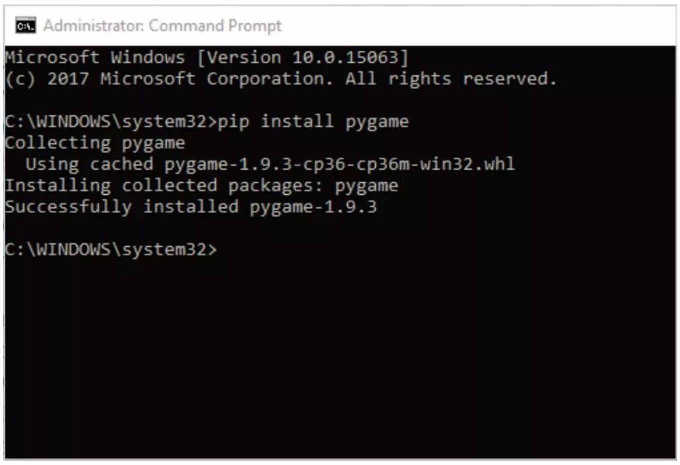
STEP 3 The result is an error in the IDLE Shell, as the pygame module isn't recognised or installed in Python. To install a module we can use PIP (Pip Installs Packages). Close down the IDLE Shell and drop into a command prompt or Terminal session. At an elevated admin command prompt, enter:

```
pip install pygame
```



```
C:\Users\david>pip install pygame
```

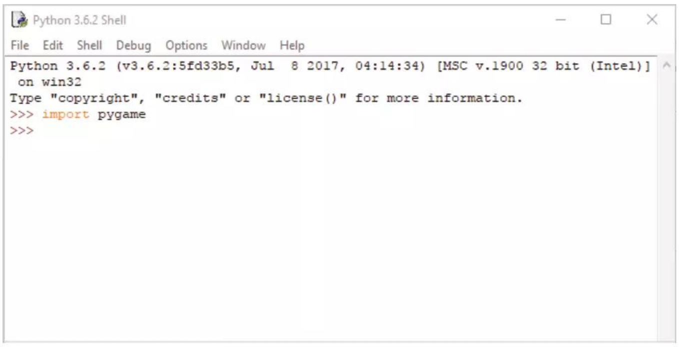
STEP 4 The PIP installation requires an elevated status due it installing components at different locations. Windows users can search for CMD via the Start button and right-click the result then click Run as Administrator. Linux and Mac users can use the Sudo command, with sudo pip install package.



```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>pip install pygame
Collecting pygame
  Using cached pygame-1.9.3-cp36-cp36m-win32.whl
Installing collected packages: pygame
Successfully installed pygame-1.9.3

C:\WINDOWS\system32>
```

STEP 5

Close the command prompt or Terminal and relaunch the IDLE Shell. When you now enter: `import pygame`, the module will be imported into the code without any problems. You'll find that most code downloaded or copied from the Internet will contain a module, mainstream of unique, these are usually the source of errors in execution due to them being missing.


Python 3.6.2 Shell
File Edit Shell Debug Options Window Help

```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> import pygame
>>>
```

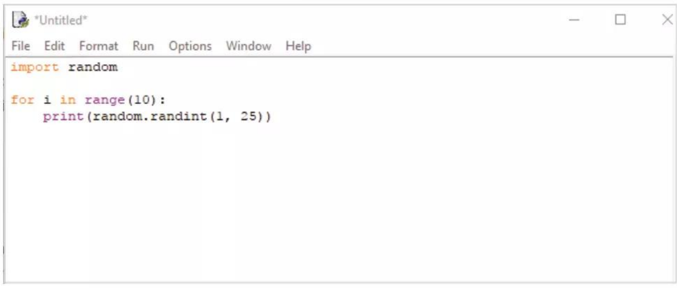
STEP 6

The modules contain the extra code needed to achieve a certain result within your own code, as we've previously experimented with. For example:

```
import random
```

Brings in the code from the random number generator module. You can then use this module to create something like:

```
for i in range(10):
  print(random.randint(1, 25))
```

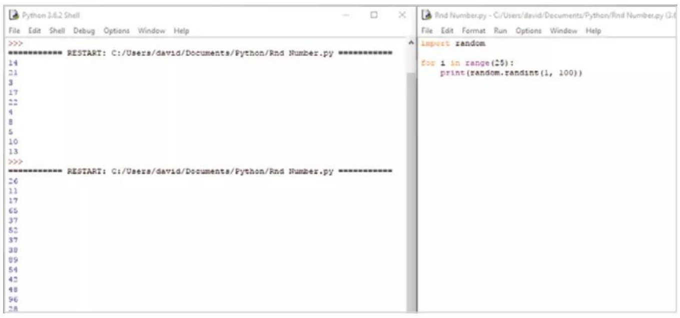

Untitled
File Edit Format Run Options Window Help

```
import random

for i in range(10):
    print(random.randint(1, 25))
```

STEP 7

This code, when saved and executed, will display ten random numbers from 1 to 25. You can play around with the code to display more or less, and from a great or lesser range. For example:

```
import random

for i in range(25):
  print(random.randint(1, 100))
```


Python 3.6.2 Shell
File Edit Shell Debug Options Window Help
Rnd Number.py - C:/Users/david/Documents/Python/Rnd Number.py (3.6
File Edit Format Run Options Window Help

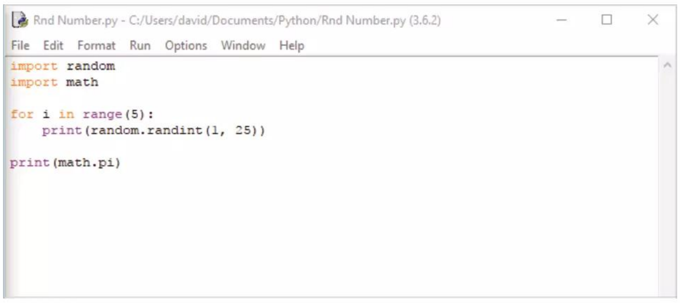
```
>>>
=========== RESTART: C:/Users/david/Documents/Python/Rnd Number.py ===========
14
21
3
17
22
4
8
5
10
13
>>>
=========== RESTART: C:/Users/david/Documents/Python/Rnd Number.py ===========
26
11
17
65
37
52
37
38
89
54
42
48
96
28
```

```
import random

for i in range(25):
    print(random.randint(1, 100))
```

STEP 8

Multiple modules can be imported within your code. To extend our example, use:

```
import random
import math

for I in range(5):
  print(random.randint(1, 25))

print(math.pi)
```

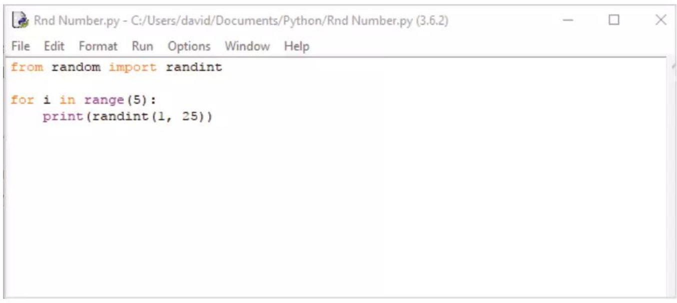

Rnd Number.py - C:/Users/david/Documents/Python/Rnd Number.py (3.6.2)
File Edit Format Run Options Window Help

```
import random
import math

for i in range(5):
    print(random.randint(1, 25))

print(math.pi)
```

STEP 9

The result is a string of random numbers followed by the value of Pi as pulled from the math module using the print(math.pi) function. You can also pull in certain functions from a module by using the from and import commands, such as:

```
from random import randint

for i in range(5):
  print(randint(1, 25))
```


Rnd Number.py - C:/Users/david/Documents/Python/Rnd Number.py (3.6.2)
File Edit Format Run Options Window Help

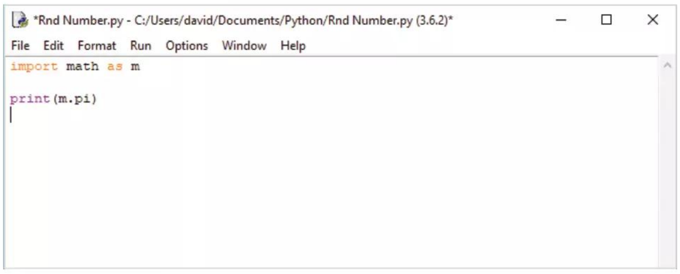
```
from random import randint

for i in range(5):
    print(randint(1, 25))
```

STEP 10

This helps create a more streamlined approach to programming. You can also use import module*, which will import everything defined within the named module. However, it's often regarded as a waste of resources but it works nonetheless. Finally, modules can be imported as aliases:

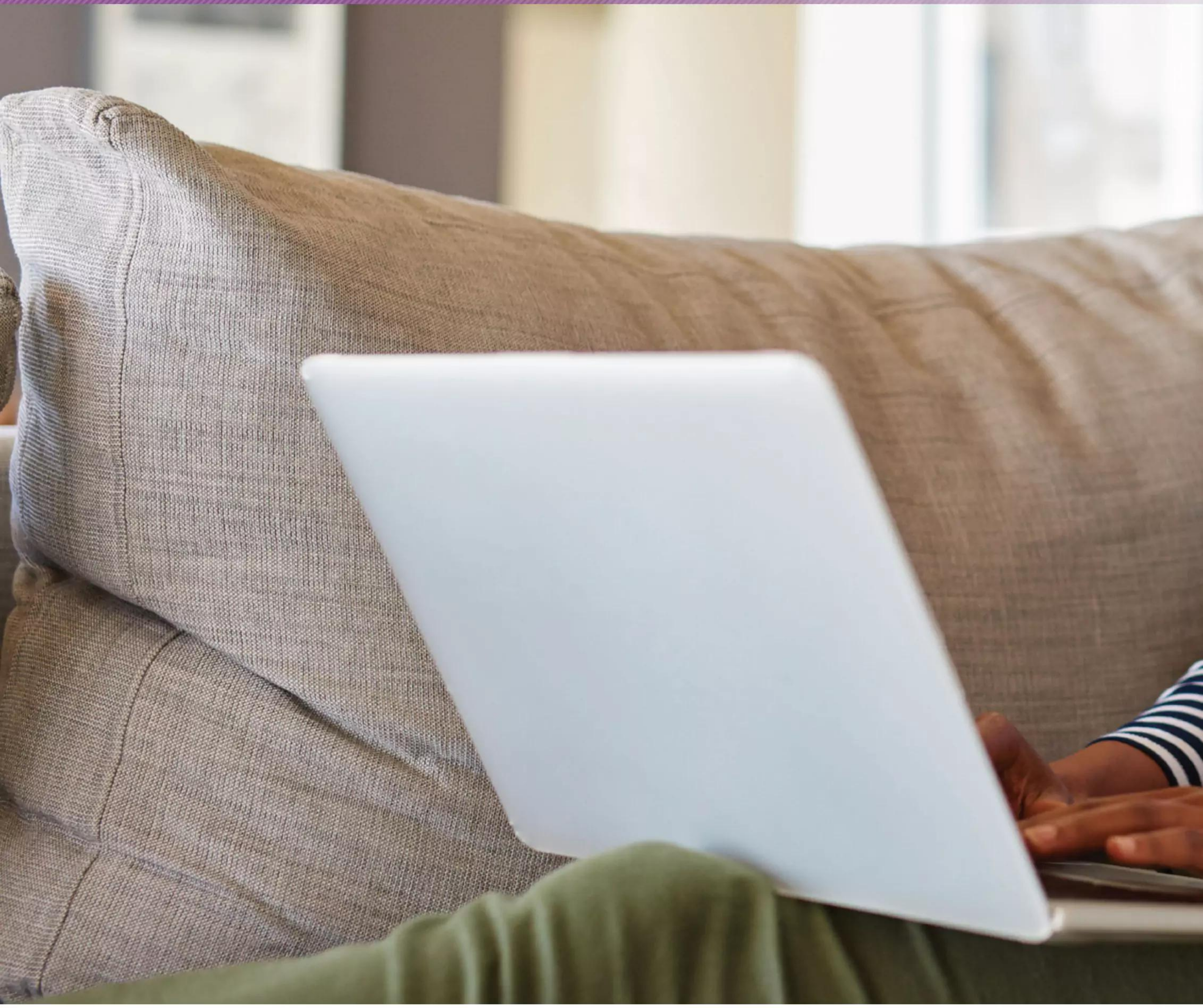
```
import math as m

print(m.pi)
```

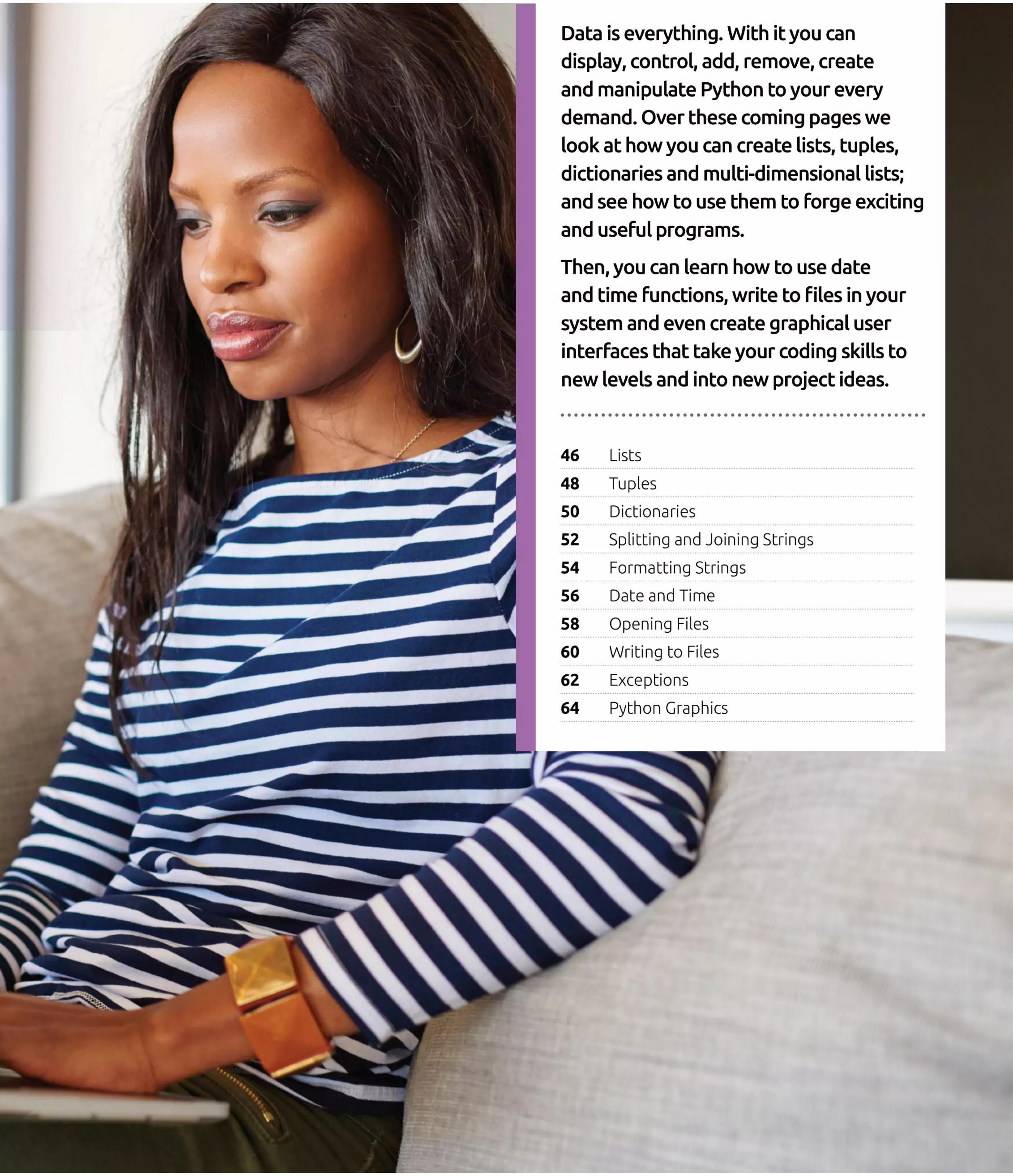
Of course, adding comments helps to tell others what's going on.


Rnd Number.py - C:/Users/david/Documents/Python/Rnd Number.py (3.6.2)
File Edit Format Run Options Window Help

```
import math as m

print(m.pi)
```

Working with Data





Data is everything. With it you can display, control, add, remove, create and manipulate Python to your every demand. Over these coming pages we look at how you can create lists, tuples, dictionaries and multi-dimensional lists; and see how to use them to forge exciting and useful programs.

Then, you can learn how to use date and time functions, write to files in your system and even create graphical user interfaces that take your coding skills to new levels and into new project ideas.

46	Lists
48	Tuples
50	Dictionaries
52	Splitting and Joining Strings
54	Formatting Strings
56	Date and Time
58	Opening Files
60	Writing to Files
62	Exceptions
64	Python Graphics

Lists

Lists are one of the most common types of data structures you will comes across in Python. A list is simply a collection of items, or data if you prefer, that can be accessed as a whole, or individually if wanted.

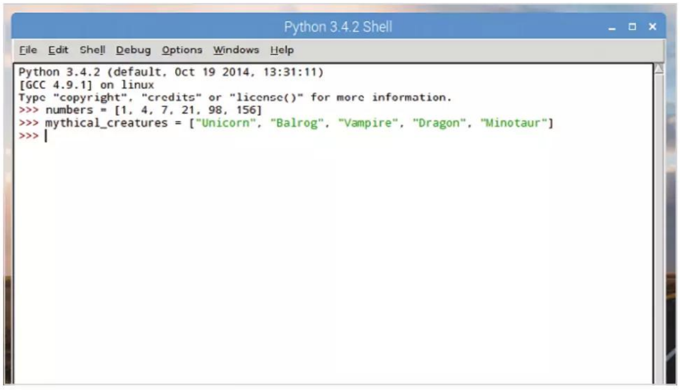
WORKING WITH LISTS

Lists are extremely handy in Python. A list can be strings, integers and also variables. You can even include functions in lists, and lists within lists.

STEP 1

A list is a sequence of data values called items. You create the name of your list followed by an equals sign, then square brackets and the items separated by commas; note that strings use quotes:

```
numbers = [1, 4, 7, 21, 98, 156]
mythical_creatures – [“Unicorn”, “Balrog”,
“Vampire”, “Dragon”, “Minotaur”]
```



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> numbers = [1, 4, 7, 21, 98, 156]
>>> mythical_creatures = ["Unicorn", "Balrog", "Vampire", "Dragon", "Minotaur"]
>>> |
```

STEP 2

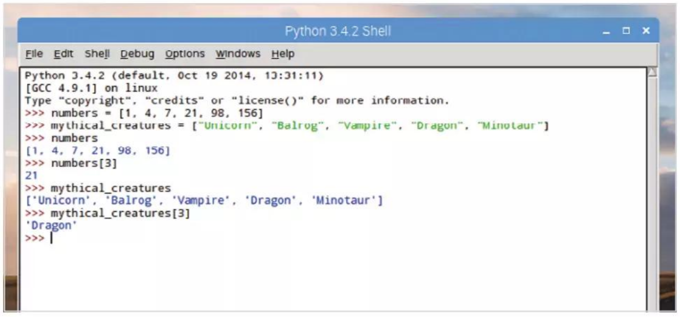
Once you've defined your list you can call each by referencing its name, followed by a number. Lists start the first item entry as 0, followed by 1, 2, 3 and so on. For example:

```
numbers
```

To call up the entire contents of the list.

```
numbers[3]
```

To call the third from zero item in the list (21 in this case).

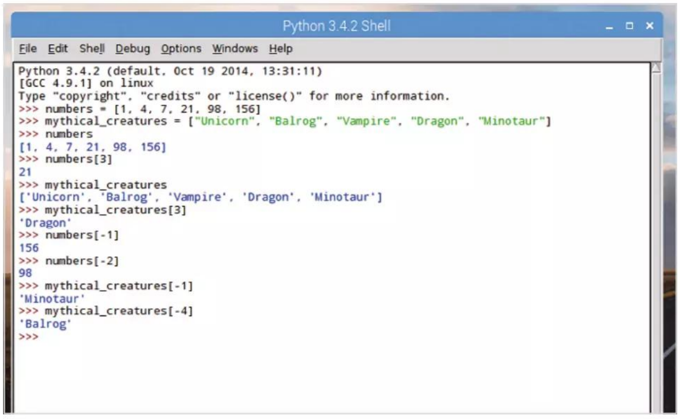


```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> numbers = [1, 4, 7, 21, 98, 156]
>>> mythical_creatures = ["Unicorn", "Balrog", "Vampire", "Dragon", "Minotaur"]
>>> numbers
[1, 4, 7, 21, 98, 156]
>>> numbers[3]
21
>>> mythical_creatures
['Unicorn', 'Balrog', 'Vampire', 'Dragon', 'Minotaur']
>>> mythical_creatures[3]
'Dragon'
>>> |
```

STEP 3

You can also access, or index, the last item in a list by using the minus sign before the item number [-1], or the second to last item with [-2] and so on. Trying to reference an item that isn't in the list, such as [10] will return an error:

```
numbers[-1]
mythical_creatures[-4]
```



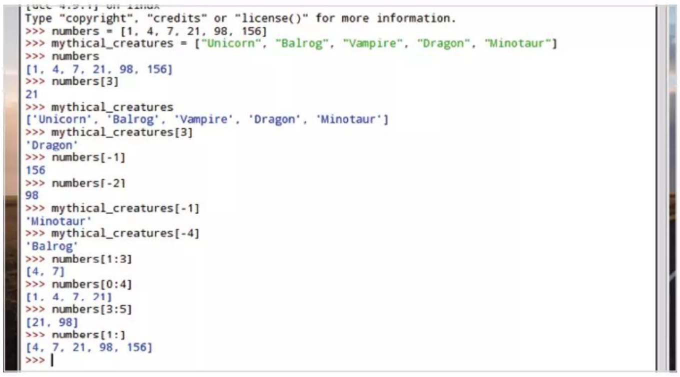
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> numbers = [1, 4, 7, 21, 98, 156]
>>> mythical_creatures = ["Unicorn", "Balrog", "Vampire", "Dragon", "Minotaur"]
>>> numbers
[1, 4, 7, 21, 98, 156]
>>> numbers[3]
21
>>> mythical_creatures
['Unicorn', 'Balrog', 'Vampire', 'Dragon', 'Minotaur']
>>> mythical_creatures[3]
'Dragon'
>>> numbers[-1]
156
>>> numbers[-2]
98
>>> mythical_creatures[-1]
'Minotaur'
>>> mythical_creatures[-4]
'Balrog'
>>>
```

STEP 4

Slicing is similar to indexing but you can retrieve multiple items in a list by separating item numbers with a colon. For example:

```
numbers[1:3]
```

Will output the 4 and 7, being item numbers 1 and 2. Note that the returned values don't include the second index position (as you would numbers[1:3] to return 4, 7 and 21).



```
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> numbers = [1, 4, 7, 21, 98, 156]
>>> mythical_creatures = ["Unicorn", "Balrog", "Vampire", "Dragon", "Minotaur"]
>>> numbers
[1, 4, 7, 21, 98, 156]
>>> numbers[3]
21
>>> mythical_creatures
['Unicorn', 'Balrog', 'Vampire', 'Dragon', 'Minotaur']
>>> mythical_creatures[3]
'Dragon'
>>> numbers[-1]
156
>>> numbers[-2]
98
>>> mythical_creatures[-1]
'Minotaur'
>>> mythical_creatures[-4]
'Balrog'
>>> numbers[1:3]
[4, 7]
>>> numbers[0:4]
[1, 4, 7, 21]
>>> numbers[3:5]
[21, 98]
>>> numbers[1:]
[4, 7, 21, 98, 156]
>>> |
```

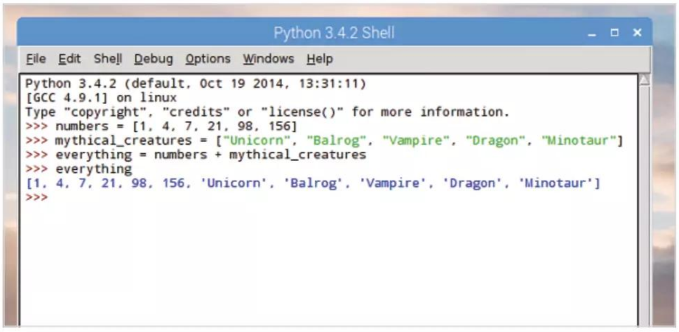
STEP 5

You can update items within an existing list, remove items and even join lists together. For example, to join two lists you can use:

```
everything = numbers + mythical_creatures
```

Then view the combined list with:

```
everything
```



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> numbers = [1, 4, 7, 21, 98, 156]
>>> mythical_creatures = ["Unicorn", "Balrog", "Vampire", "Dragon", "Minotaur"]
>>> everything = numbers + mythical_creatures
>>> everything
[1, 4, 7, 21, 98, 156, 'Unicorn', 'Balrog', 'Vampire', 'Dragon', 'Minotaur']
>>>
```

STEP 6

Items can be added to a list by entering:

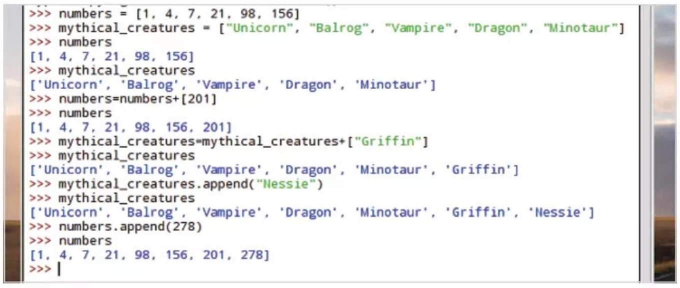
```
numbers=numbers+[201]
```

Or for strings:

```
mythical_creatres=mythical_creatures+[“Griffin”]
```

Or by using the append function:

```
mythical_creatures.append(“Nessie”)
numbers.append(278)
```



```
>>> numbers = [1, 4, 7, 21, 98, 156]
>>> mythical_creatures = ["Unicorn", "Balrog", "Vampire", "Dragon", "Minotaur"]
>>> numbers
[1, 4, 7, 21, 98, 156]
>>> mythical_creatures
['Unicorn', 'Balrog', 'Vampire', 'Dragon', 'Minotaur']
>>> numbers=numbers+[201]
>>> numbers
[1, 4, 7, 21, 98, 156, 201]
>>> mythical_creatures=mythical_creatures+["Griffin"]
>>> mythical_creatures
['Unicorn', 'Balrog', 'Vampire', 'Dragon', 'Minotaur', 'Griffin']
>>> mythical_creatures.append("Nessie")
>>> mythical_creatures
['Unicorn', 'Balrog', 'Vampire', 'Dragon', 'Minotaur', 'Griffin', 'Nessie']
>>> numbers.append(278)
>>> numbers
[1, 4, 7, 21, 98, 156, 201, 278]
>>> |
```

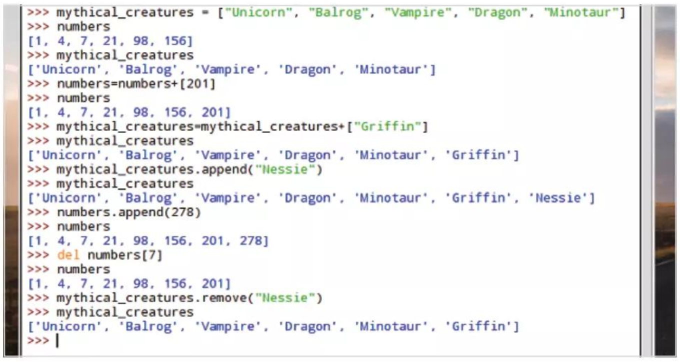
STEP 7

Removal of items can be done in two ways. The first is by the item number:

```
del numbers[7]
```

Alternatively, by item name:

```
mythical_creatures.remove(“Nessie”)
```



```
>>> mythical_creatures = ["Unicorn", "Balrog", "Vampire", "Dragon", "Minotaur"]
>>> numbers
[1, 4, 7, 21, 98, 156]
>>> mythical_creatures
['Unicorn', 'Balrog', 'Vampire', 'Dragon', 'Minotaur']
>>> numbers=numbers+[201]
>>> numbers
[1, 4, 7, 21, 98, 156, 201]
>>> mythical_creatures=mythical_creatures+["Griffin"]
>>> mythical_creatures
['Unicorn', 'Balrog', 'Vampire', 'Dragon', 'Minotaur', 'Griffin']
>>> mythical_creatures.append("Nessie")
>>> mythical_creatures
['Unicorn', 'Balrog', 'Vampire', 'Dragon', 'Minotaur', 'Griffin', 'Nessie']
>>> numbers.append(278)
>>> numbers
[1, 4, 7, 21, 98, 156, 201, 278]
>>> del numbers[7]
>>> numbers
[1, 4, 7, 21, 98, 156, 201]
>>> mythical_creatures.remove("Nessie")
>>> mythical_creatures
['Unicorn', 'Balrog', 'Vampire', 'Dragon', 'Minotaur', 'Griffin']
>>> |
```

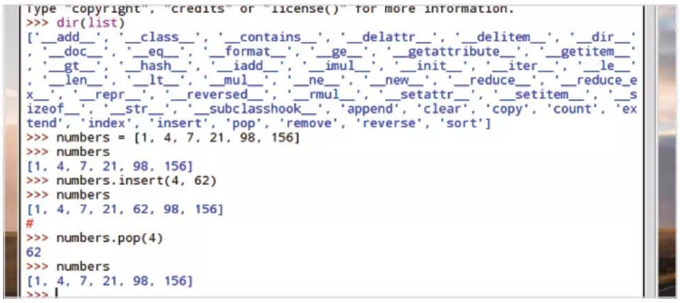
STEP 8

You can view what can be done with lists by entering dir(list) into the Shell. The output is the available functions, for example, insert and pop are used to add and remove items at certain positions. To insert the number 62 at item index 4:

```
numbers.insert(4, 62)
```

To remove it:

```
numbers.pop(4)
```



```
Type "copyright", "credits" or "license()" for more information.
>>> dir(list)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__'
, '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__'
, '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__iter__', '__le__'
, '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_e
x__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__s
izeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'ex
tend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
>>> numbers = [1, 4, 7, 21, 98, 156]
>>> numbers
[1, 4, 7, 21, 98, 156]
>>> numbers.insert(4, 62)
>>> numbers
[1, 4, 7, 21, 62, 98, 156]
#
>>> numbers.pop(4)
62
>>> numbers
[1, 4, 7, 21, 98, 156]
>>> |
```

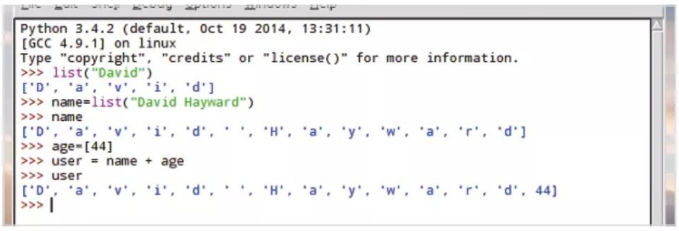
STEP 9

You also use the list function to break a string down into its components. For example:

```
list(“David”)
```

Breaks the name David into ‘D’, ‘a’, ‘v’, ‘i’, ‘d’. This can then be passed to a new list:

```
name=list(“David Hayward”)
name
age=[44]
user = name + age
user
```



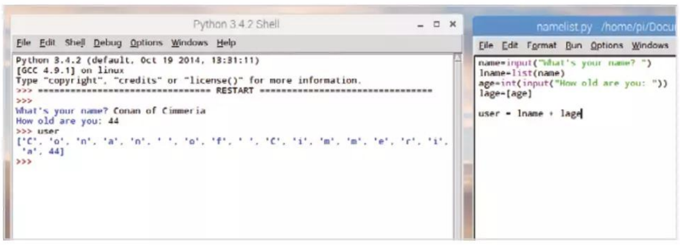
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> list("David")
['D', 'a', 'v', 'i', 'd']
>>> name=list("David Hayward")
>>> name
['D', 'a', 'v', 'i', 'd', ' ', 'H', 'a', 'y', 'w', 'a', 'r', 'd']
>>> age=[44]
>>> user = name + age
>>> user
['D', 'a', 'v', 'i', 'd', ' ', 'H', 'a', 'y', 'w', 'a', 'r', 'd', 44]
>>> |
```

STEP 10

Based on that, you can create a program to store someone’s name and age as a list:

```
name=input(“What’s your name? “)
lname=list(name)
age=int(input(“How old are you: “))
lage=[age]
user = lname + lage
```

The combined name and age list is called user, which can be called by entering user into the Shell. Experiment and see what you can do.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
namelist.py /home/pi/Docu
File Edit Format Run Options Windows

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
What's your name? Conan of Cimmeria
How old are you: 44
>>> user
['C', 'o', 'n', 'a', 'n', ' ', 'o', 'f', ' ', 'C', 'i', 'm', 'm', 'e', 'r', 'i',
 'a', 44]
>>>
```

```
name=input("What's your name? ")
lname=list(name)
age=int(input("How old are you: "))
lage=[age]

user = lname + lage
```

Tuples

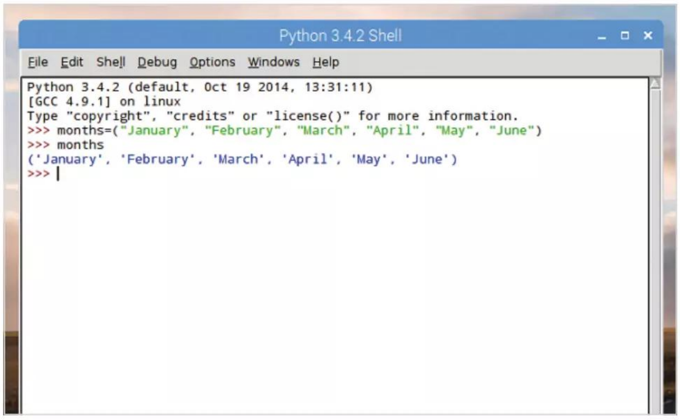
Tuples are very much identical to lists. However, where lists can be updated, deleted or changed in some way, a tuple remains a constant. This is called immutable and they're perfect for storing fixed data items.

THE IMMUTABLE TUPLE

Reasons for having tuples vary depending on what the program is intended to do. Normally, a tuple is reserved for something special but they're also used for example, in an adventure game, where non-playing character names are stored.

STEP 1 A tuple is created the same way as a list but in this instance you use curved brackets instead of square brackets. For example:

```
months=("January", "February", "March", "April",
"May", "June")
months
```

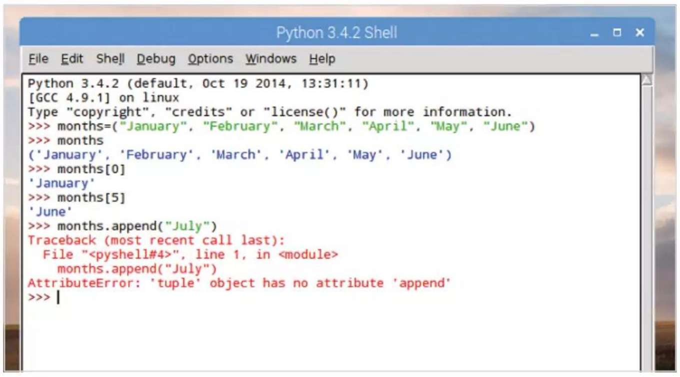


```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> months=("January", "February", "March", "April", "May", "June")
>>> months
('January', 'February', 'March', 'April', 'May', 'June')
>>> |
```

STEP 2 Just as with lists, the items within a named tuple can be indexed according to their position in the data range, i.e.:

```
months[0]
months[5]
```

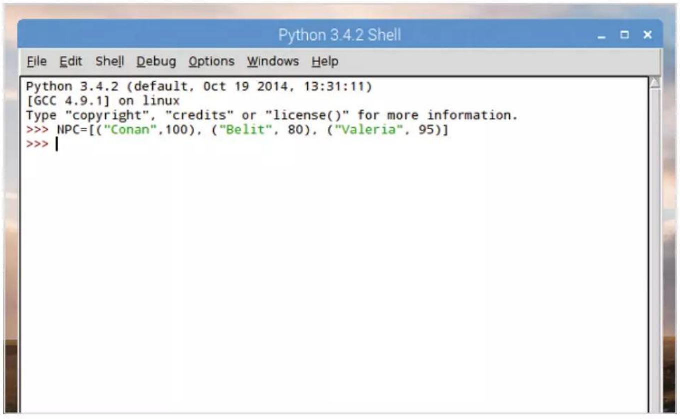
However, any attempt at deleting or adding to the tuple will result in an error in the Shell.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> months=("January", "February", "March", "April", "May", "June")
>>> months
('January', 'February', 'March', 'April', 'May', 'June')
>>> months[0]
'January'
>>> months[5]
'June'
>>> months.append("July")
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    months.append("July")
AttributeError: 'tuple' object has no attribute 'append'
>>> |
```

STEP 3 You can create grouped tuples into lists that contain multiple sets of data. For instance, here is a tuple called NPC (Non-Playable Characters) containing the character name and their combat rating for an adventure game:

```
NPC=[("Conan", 100), ("Belit", 80), ("Valeria",
95)]
```

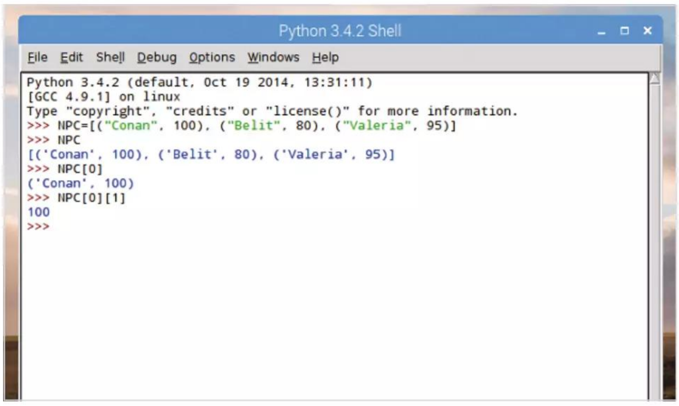


```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> NPC=[("Conan",100), ("Belit", 80), ("Valeria", 95)]
>>> |
```

STEP 4 Each of these data items can be accessed as a whole by entering NPC into the Shell; or they can be indexed according to their position NPC[0]. You can also index the individual tuples within the NPC list:

```
NPC[0][1]
```

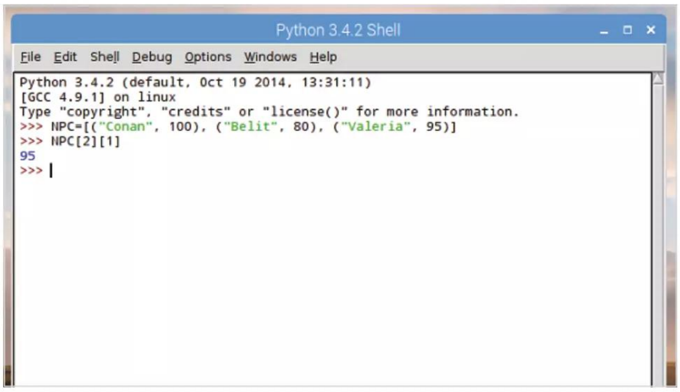
Will display 100.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> NPC=[("Conan", 100), ("Belit", 80), ("Valeria", 95)]
>>> NPC
[('Conan', 100), ('Belit', 80), ('Valeria', 95)]
>>> NPC[0]
('Conan', 100)
>>> NPC[0][1]
100
>>>
```

STEP 5 It's worth noting that when referencing multiple tuples within a list, the indexing is slightly different from the norm. You would expect the 95 combat rating of the character Valeria to be NPC[4][5], but it's not. It's actually:

```
NPC[2][1]
```

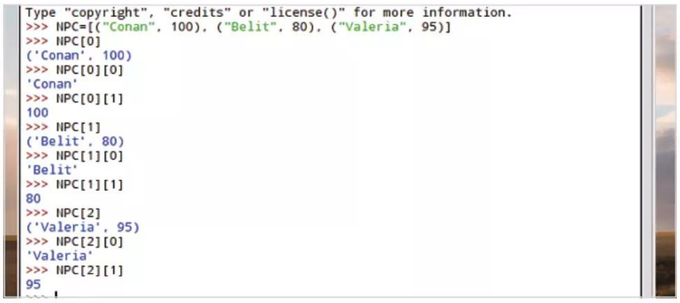

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> NPC=[("Conan", 100), ("Belit", 80), ("Valeria", 95)]
>>> NPC[2][1]
95
>>> |
```

STEP 6 This means of course that the indexing follows thus:

```
0
0, 0
0, 1
1
1, 0
1, 1
2
2, 0
2,1
```

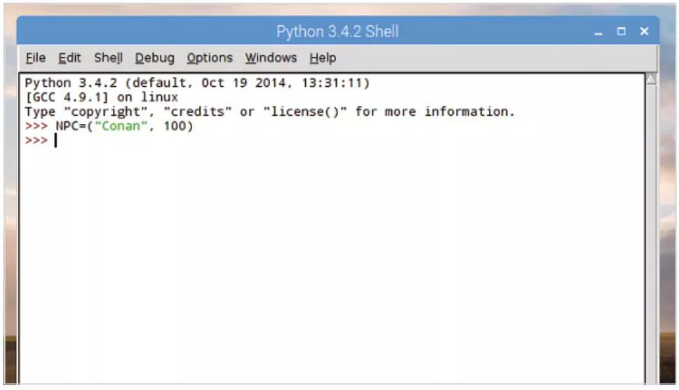
Which as you can imagine, gets a little confusing when you've got a lot of tuple data to deal with.



```
Type "copyright", "credits" or "license()" for more information.
>>> NPC=[("Conan", 100), ("Belit", 80), ("Valeria", 95)]
>>> NPC[0]
('Conan', 100)
>>> NPC[0][0]
'Conan'
>>> NPC[0][1]
100
>>> NPC[1]
('Belit', 80)
>>> NPC[1][0]
'Belit'
>>> NPC[1][1]
80
>>> NPC[2]
('Valeria', 95)
>>> NPC[2][0]
'Valeria'
>>> NPC[2][1]
95
```

STEP 7 Tuples though utilise a feature called unpacking, where the data items stored within a tuple are assigned variables. First create the tuple with two items (name and combat rating):

```
NPC=("Conan", 100)
```

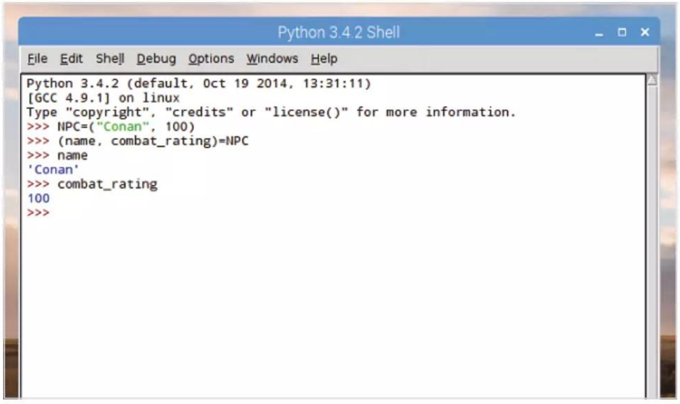

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> NPC=("Conan", 100)
>>> |
```

STEP 8 Now unpack the tuple into two corresponding variables:

```
(name, combat_rating)=NPC
```

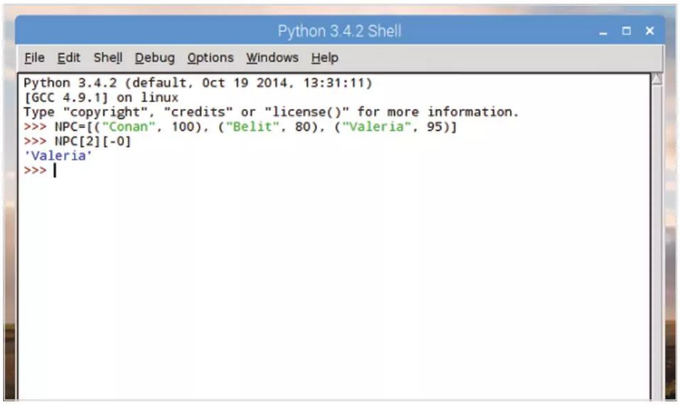
You can now check the values by entering name and combat_rating.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> NPC=("Conan", 100)
>>> (name, combat_rating)=NPC
>>> name
'Conan'
>>> combat_rating
100
>>>
```

STEP 9 Remember, as with lists, you can also index tuples using negative numbers which count backwards from the end of the data list. For our example, using the tuple with multiple data items, you would reference the Valeria character with:

```
NPC[2][-0]
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

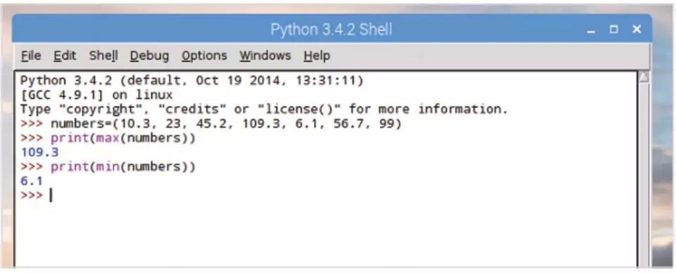
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> NPC=[("Conan", 100), ("Belit", 80), ("Valeria", 95)]
>>> NPC[2][-0]
'Valeria'
>>> |
```

STEP 10 You can use the max and min functions to find the highest and lowest values of a tuple composed of numbers. For example:

```
numbers=(10.3, 23, 45.2, 109.3, 6.1, 56.7, 99)
```

The numbers can be integers and floats. To output the highest and lowest, use:

```
print(max(numbers))
print(min(numbers))
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> numbers=(10.3, 23, 45.2, 109.3, 6.1, 56.7, 99)
>>> print(max(numbers))
109.3
>>> print(min(numbers))
6.1
>>> |
```

Dictionaries

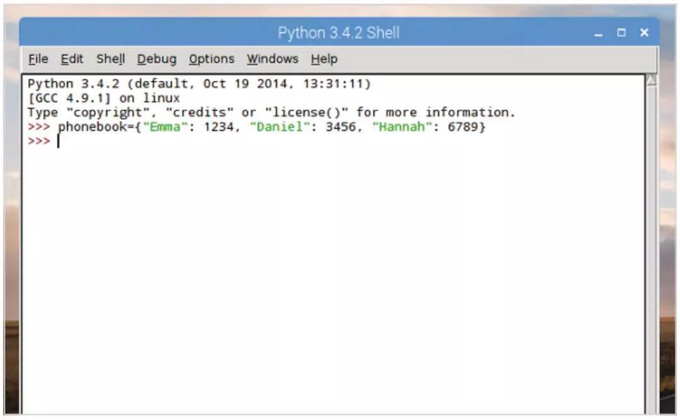
Lists are extremely useful but dictionaries in Python are by far the more technical way of dealing with data items. They can be tricky to get to grips with at first but you'll soon be able to apply them to your own code.

KEY PAIRS

A dictionary is like a list but instead each data item comes as a pair, these are known as Key and Value. The Key part must be unique and can either be a number or string whereas the Value can be any data item you like.

STEP 1 Let's say you want to create a phonebook in Python. You would create the dictionary name and enter the data in curly brackets, separating the key and value by a colon `Key:Value`. For example:

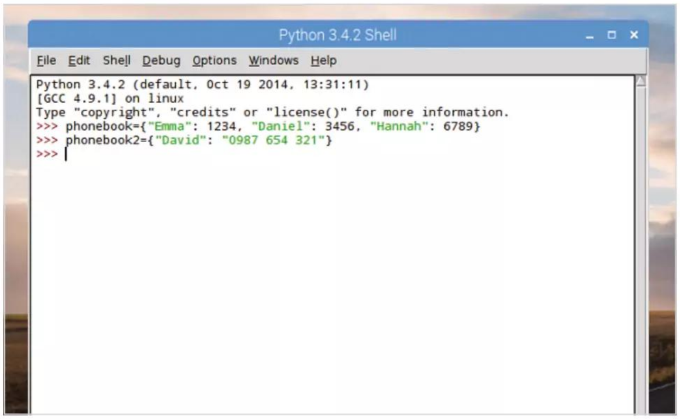
```
phonebook={“Emma”: 1234, “Daniel”: 3456, “Hannah”:
6789}
```



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> phonebook={"Emma": 1234, "Daniel": 3456, "Hannah": 6789}
>>> 
```

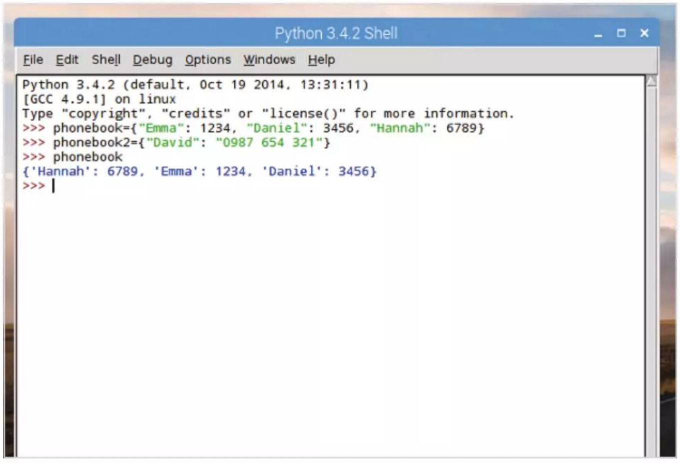
STEP 2 Just as with most lists, tuples and so on, strings need be enclosed in quotes (single or double), whilst integers can be left open. Remember that the value can be either a string or an integer, you just need to enclose the relevant one in quotes:

```
phonebook2={“David”: “0987 654 321”}
```



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> phonebook={"Emma": 1234, "Daniel": 3456, "Hannah": 6789}
>>> phonebook2={"David": "0987 654 321"}
>>> 
```

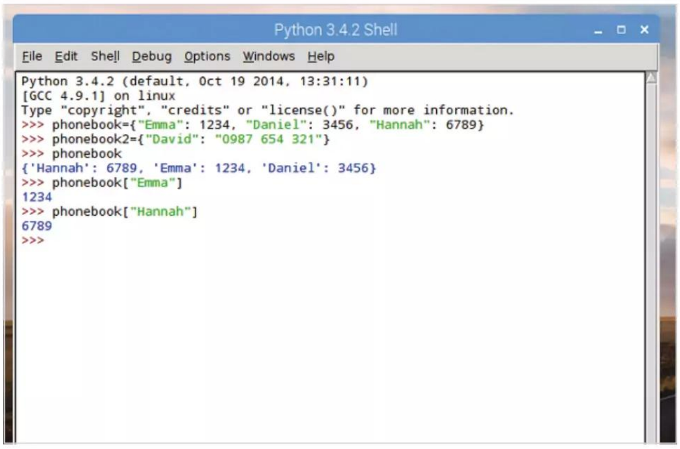
STEP 3 As with lists and tuples, you can check the contents of a dictionary by giving the dictionary a name: phonebook, in this example. This will display the data items you've entered in a similar fashion to a list, which you're no doubt familiar with by now.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> phonebook={"Emma": 1234, "Daniel": 3456, "Hannah": 6789}
>>> phonebook2={"David": "0987 654 321"}
>>> phonebook
{'Hannah': 6789, 'Emma': 1234, 'Daniel': 3456}
>>> 
```

STEP 4 The benefit of using a dictionary is that you can enter the key to index the value. Using the phonebook example from the previous steps, you can enter:

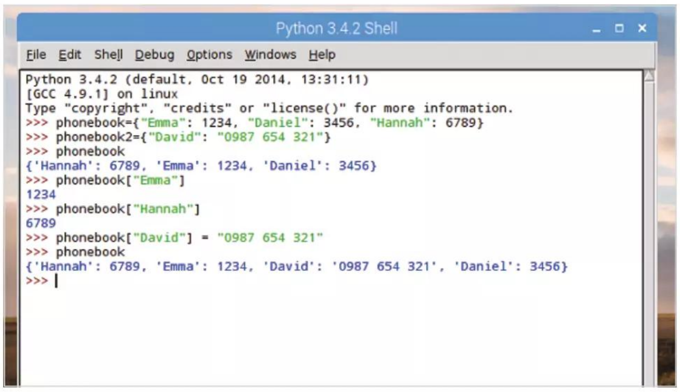
```
phonebook[“Emma”]
phonebook[“Hannah”]
```



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> phonebook={"Emma": 1234, "Daniel": 3456, "Hannah": 6789}
>>> phonebook2={"David": "0987 654 321"}
>>> phonebook
{'Hannah': 6789, 'Emma': 1234, 'Daniel': 3456}
>>> phonebook["Emma"]
1234
>>> phonebook["Hannah"]
6789
>>> 
```

STEP 5 Adding to a dictionary is easy too. You can include a new data item entry by adding the new key and value items like:

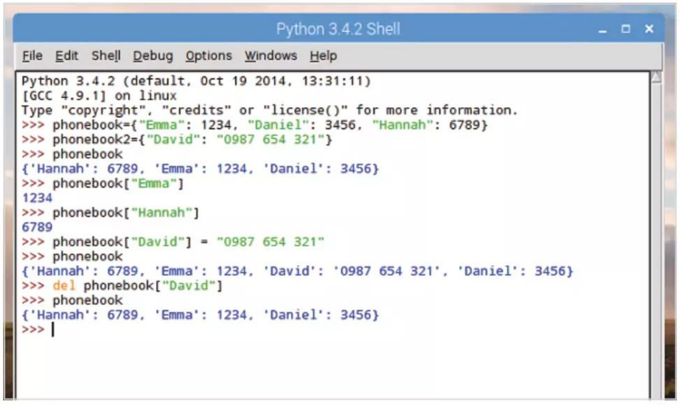
```
phonebook["David"] = "0987 654 321"
phonebook
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> phonebook={"Emma": 1234, "Daniel": 3456, "Hannah": 6789}
>>> phonebook2={"David": "0987 654 321"}
>>> phonebook
{'Hannah': 6789, 'Emma': 1234, 'Daniel': 3456}
>>> phonebook["Emma"]
1234
>>> phonebook["Hannah"]
6789
>>> phonebook["David"] = "0987 654 321"
>>> phonebook
{'Hannah': 6789, 'Emma': 1234, 'David': '0987 654 321', 'Daniel': 3456}
>>> |
```

STEP 6 You can also remove items from a dictionary by issuing the del command followed by the item's key; the value will be removed as well, since both work as a pair of data items:

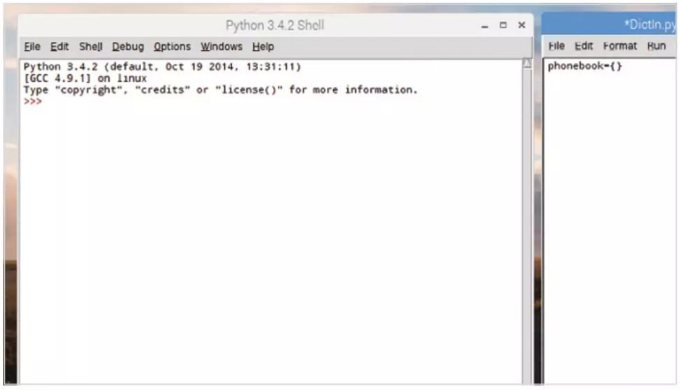
```
del phonebook["David"]
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> phonebook={"Emma": 1234, "Daniel": 3456, "Hannah": 6789}
>>> phonebook2={"David": "0987 654 321"}
>>> phonebook
{'Hannah': 6789, 'Emma': 1234, 'Daniel': 3456}
>>> phonebook["Emma"]
1234
>>> phonebook["Hannah"]
6789
>>> phonebook["David"] = "0987 654 321"
>>> phonebook
{'Hannah': 6789, 'Emma': 1234, 'David': '0987 654 321', 'Daniel': 3456}
>>> del phonebook["David"]
>>> phonebook
{'Hannah': 6789, 'Emma': 1234, 'Daniel': 3456}
>>> |
```

STEP 7 Taking this a step further, how about creating a piece of code that will ask the user for the dictionary key and value items? Create a new Editor instance and start by coding in a new, blank dictionary:

```
phonebook={}
```

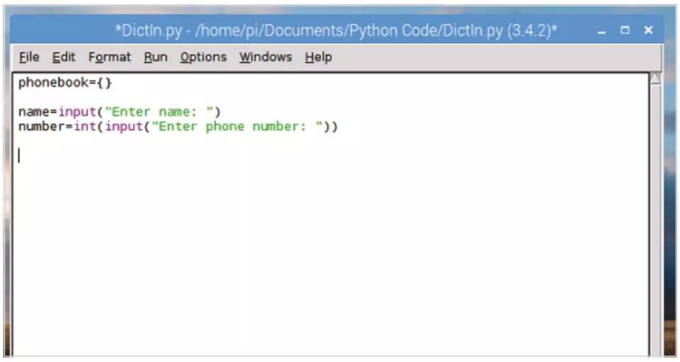

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
*Dictln.py
File Edit Format Run

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
```

```
phonebook={}
```

STEP 8 Next, you need to define the user inputs and variables: one for the person's name, the other for their phone number (let's keep it simple to avoid lengthy Python code):

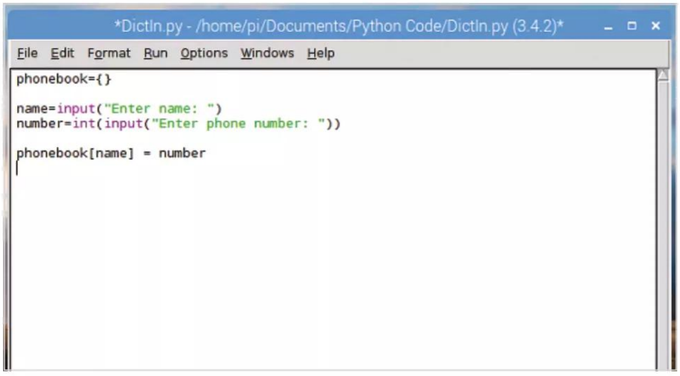
```
name=input("Enter name: ")
number=int(input("Enter phone number: "))
```


Dictln.py - /home/pi/Documents/Python Code/Dictln.py (3.4.2)
File Edit Format Run Options Windows Help

```
phonebook={}

name=input("Enter name: ")
number=int(input("Enter phone number: "))

|
```

STEP 9 Note we've kept the number as an integer instead of a string, even though the value can be both an integer or a string. Now you need to add the user's inputted variables to the newly created blank dictionary. Using the same process as in Step 5, you can enter:

```
phonebook[name] = number
```

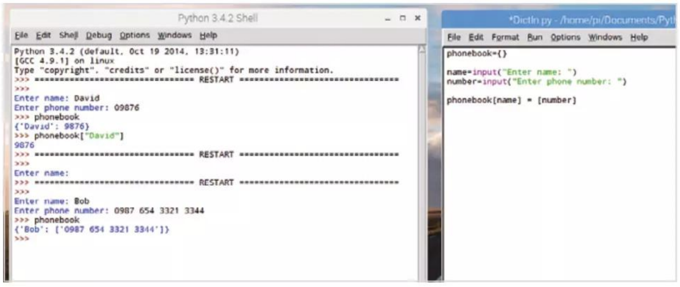

Dictln.py - /home/pi/Documents/Python Code/Dictln.py (3.4.2)
File Edit Format Run Options Windows Help

```
phonebook={}

name=input("Enter name: ")
number=int(input("Enter phone number: "))

phonebook[name] = number
|
```

STEP 10 Now when you save and execute the code, Python will ask for a name and a number. It will then insert those entries into the phonebook dictionary, which you can test by entering into the Shell:

```
phonebook
phonebook["David"]
```

If the number needs to contain spaces you need to make it a string, so remove the int part of the input.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
*Dictln.py - /home/pi/Documents/Pyt
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Enter name: David
Enter phone number: 09876
>>> phonebook
{'David': 9876}
>>> phonebook["David"]
9876
>>> ================================ RESTART ================================
>>>
Enter name:
>>> ================================ RESTART ================================
>>>
Enter name: Bob
Enter phone number: 0987 654 3321 3344
>>> phonebook
{'Bob': ['0987 654 3321 3344']}
>>>
```

```
phonebook={}

name=input("Enter name: ")
number=input("Enter phone number: ")

phonebook[name] = [number]
```

Splitting and Joining Strings

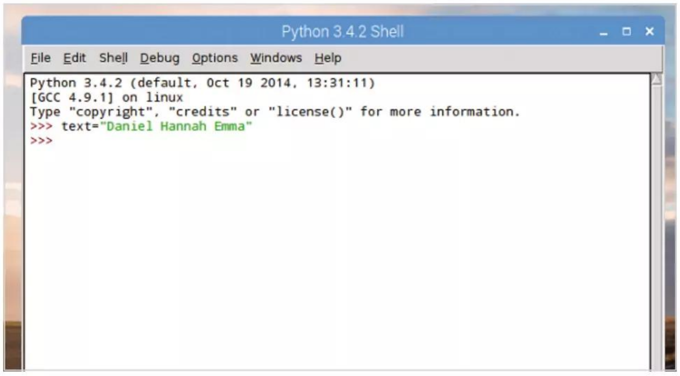
When dealing with data in Python, especially from a user's input, you will undoubtedly come across long sets of strings. A useful skill to learn in Python programming is being able to split those long strings for better readability.

STRING THEORIES

You've already looked at some list functions, using .insert, .remove, and .pop but there are also functions that can be applied to strings.

STEP 1 The main tool in the string function arsenal is .split(). With it you're able to split apart a string of data, based on the argument within the brackets. For example, here's a string with three items, each separated by a space:

```
text=”Daniel Hannah Emma”
```

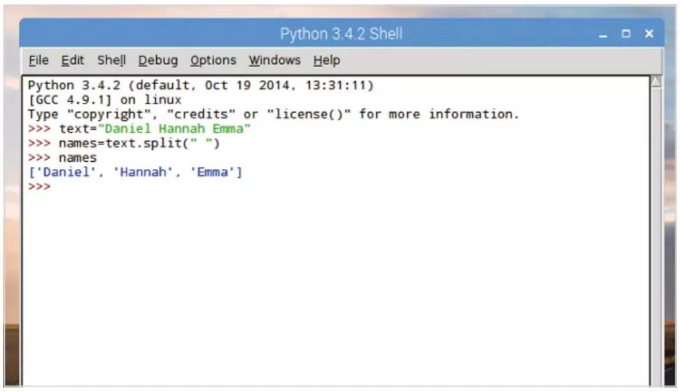

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> text="Daniel Hannah Emma"
>>>
```

STEP 2 Now let's turn the string into a list and split the content accordingly:

```
names=text.split(“ “)
```

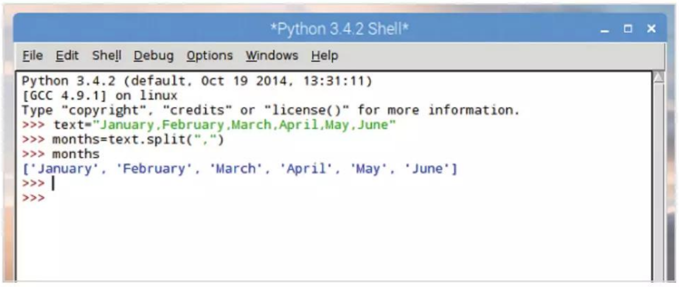
Then enter the name of the new list, names, to see the three items.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> text="Daniel Hannah Emma"
>>> names=text.split(" ")
>>> names
['Daniel', 'Hannah', 'Emma']
>>>
```

STEP 3 Note that the text.split part has the brackets, quotes, then a space followed by closing quotes and brackets. The space is the separator, indicating that each list item entry is separated by a space. Likewise, CSV (Comma Separated Value) content has a comma, so you'd use:

```
text=”January,February,March,April,May,June”
months=text.split(“,”)
months
```

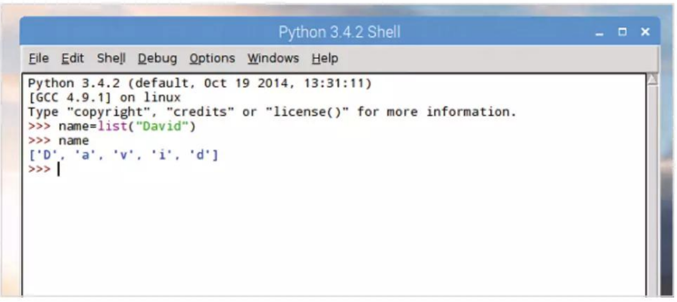

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> text="January,February,March,April,May,June"
>>> months=text.split(",")
>>> months
['January', 'February', 'March', 'April', 'May', 'June']
>>> |
>>>
```

STEP 4 You've previously seen how you can split a string into individual letters as a list, using a name:

```
name=list(“David”)
name
```

The returned value is 'D', 'a', 'v', 'i', 'd'. Whilst it may seem a little useless under ordinary circumstances, it could be handy for creating a spelling game for example.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

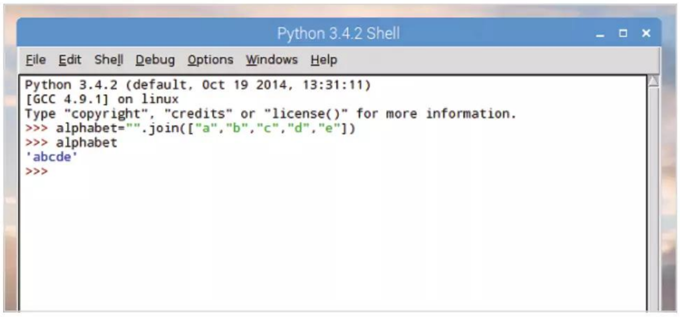
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name=list("David")
>>> name
['D', 'a', 'v', 'i', 'd']
>>> |
```

STEP 5

The opposite of the .split function is .join, where you will have separate items in a string and can join them all together to form a word or just a combination of items, depending on the program you're writing. For instance:

```
alphabet=””.join([“a”,”b”,”c”,”d”,”e”])
alphabet
```

This will display 'abcde' in the Shell.



Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

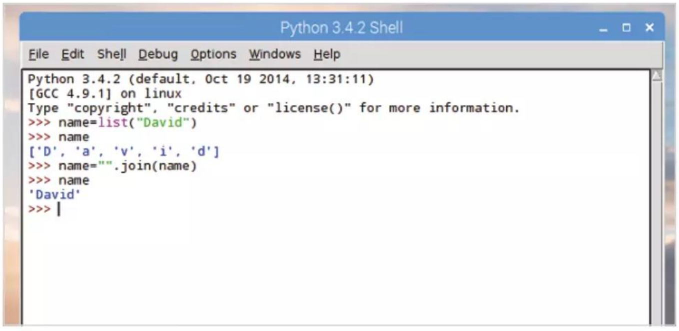
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> alphabet="".join(["a","b","c","d","e"])
>>> alphabet
'abcde'
>>>
```

STEP 6

You can therefore apply .join to the separated name you made in Step 4, combining the letters again to form the name:

```
name=””.join(name)
name
```

We've joined the string back together, and retained the list called name, passing it through the .join function.



Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

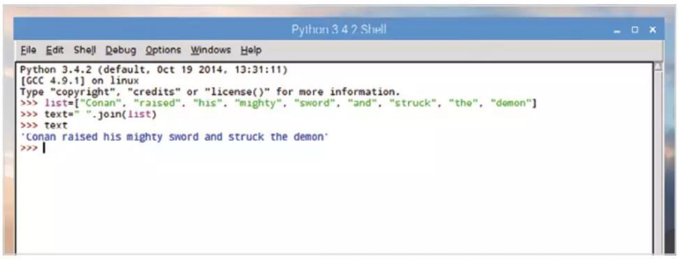
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name=list("David")
>>> name
['D', 'a', 'v', 'i', 'd']
>>> name="".join(name)
>>> name
'David'
>>> |
```

STEP 7

A good example of using the .join function is when you have a list of words you want to combine into a sentence:

```
list=[“Conan”, “raised”, “his”, “mighty”, “sword”,
“and”, “struck”, “the”, “demon”]
text=” “.join(list)
text
```

Note the space between the quotes before the .join function (where there were no spaces in the Step 6's join)



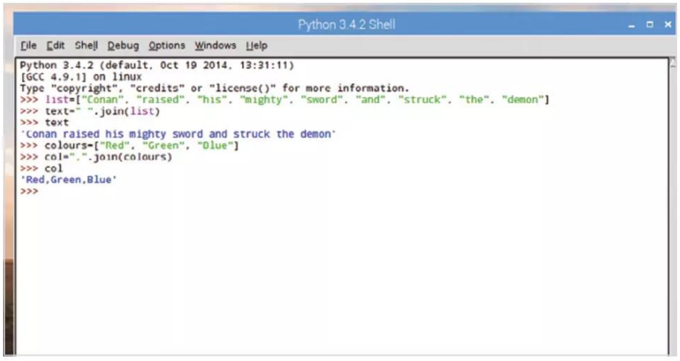
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> list=["Conan", "raised", "his", "mighty", "sword", "and", "struck", "the", "demon"]
>>> text=" ".join(list)
>>> text
'Conan raised his mighty sword and struck the demon'
>>> |
```

STEP 8

As with the .split function, the separator doesn't have to be a space, it can also be a comma, a full stop, a hyphen or whatever you like:

```
colours=[“Red”, “Green”, “Blue”]
col=”,”.join(colours)
col
```



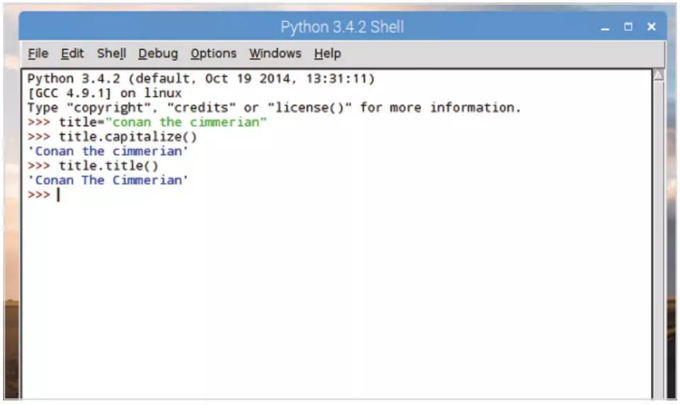
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> list=["Conan", "raised", "his", "mighty", "sword", "and", "struck", "the", "demon"]
>>> text=" ".join(list)
>>> text
'Conan raised his mighty sword and struck the demon'
>>> colours=["Red", "Green", "Blue"]
>>> col=",".join(colours)
>>> col
'Red,Green,Blue'
>>>
```

STEP 9

There's some interesting functions you apply to a string, such as .capitalize and .title. For example:

```
title=”conan the cimmerian”
title.capitalize()
title.title()
```



Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

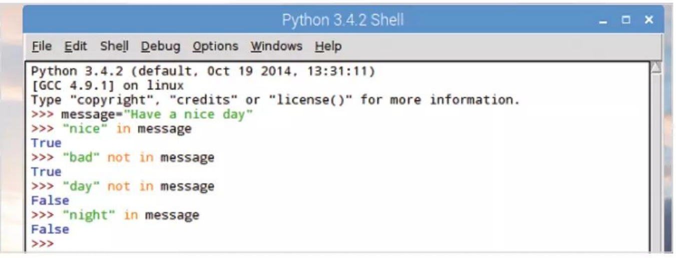
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> title="conan the cimmerian"
>>> title.capitalize()
'Conan the cimmerian'
>>> title.title()
'Conan The Cimmerian'
>>> |
```

STEP 10

You can also use logic operators on strings, with the 'in' and 'not in' functions. These enable you to check if a string contains (or does not contain) a sequence of characters:

```
message=”Have a nice day”
“nice” in message
```

```
”bad” not in message
“day” not in message
“night” in message
```



Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> message="Have a nice day"
>>> "nice" in message
True
>>> "bad" not in message
True
>>> "day" not in message
False
>>> "night" in message
False
>>>
```

Formatting Strings

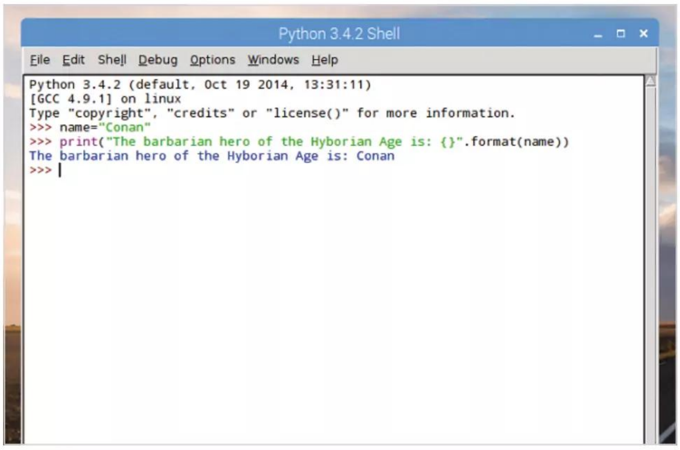
When you work with data, creating lists, dictionaries and objects you may often want to print out the results. Merging strings with data is easy especially with Python 3, as earlier versions of Python tended to complicate matters.

STRING FORMATTING

Since Python 3, string formatting has become a much neater process, using the .format function combined with curly brackets. It's a more logical and better formed approach than previous versions.

STEP 1 The basic formatting in Python is to call each variable into the string using the curly brackets:

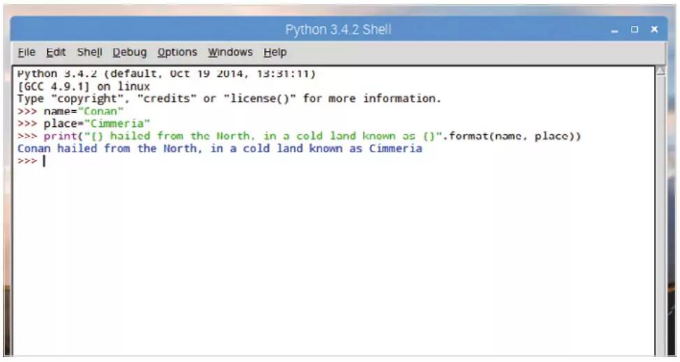
```
name=”Conan”
print(“The barbarian hero of the Hyborian Age is:
{}”.format(name))
```



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name="Conan"
>>> print("The barbarian hero of the Hyborian Age is: {}".format(name))
The barbarian hero of the Hyborian Age is: Conan
>>> |
```

STEP 2 Remember to close the print function with two sets of brackets, as you've encased the variable in one, and the print function in another. You can include multiple cases of string formatting in a single print function:

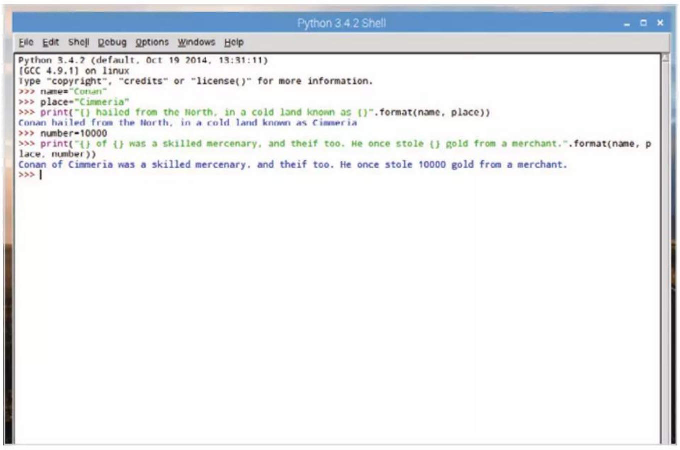
```
name=”Conan”
place=”Cimmeria”
print(“{} hailed from the North, in a cold land
known as {}”.format(name, place))
```



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name="Conan"
>>> place="Cimmeria"
>>> print("{} hailed from the North, in a cold land known as {}".format(name, place))
Conan hailed from the North, in a cold land known as Cimmeria
>>> |
```

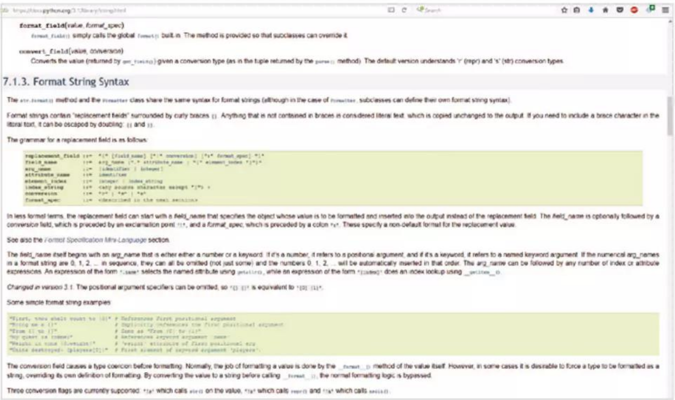
STEP 3 You can of course also include integers into the mix:

```
number=10000
print(“{} of {} was a skilled mercenary,
and thief too. He once stole {} gold from a
merchant.”.format(name, place, number))
```



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name="Conan"
>>> place="Cimmeria"
>>> print("{} hailed from the North, in a cold land known as {}".format(name, place))
Conan hailed from the North, in a cold land known as Cimmeria
>>> number=10000
>>> print("{} of {} was a skilled mercenary, and theif too. He once stole {} gold from a merchant.".format(name, p
lace, number))
Conan of Cimmeria was a skilled mercenary, and theif too. He once stole 10000 gold from a merchant.
>>> |
```

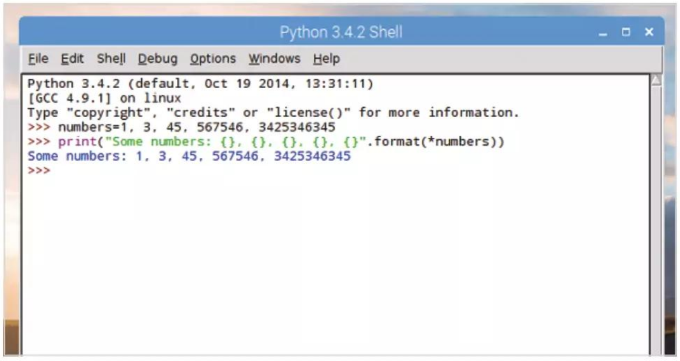
STEP 4 There are many different ways to apply string formatting, some are quite simple, as we've shown you here; others can be significantly more complex. It all depends on what you want from your program. A good place to reference frequently regarding string formatting is the Python Docs webpage, found at **www.docs.python.org/3.1/library/string.html**. Here, you will find tons of help.



STEP 5

Interestingly you can reference a list using the string formatting function. You need to place an asterisk in front of the list name:

```
numbers=1, 3, 45, 567546, 3425346345
print("Some numbers: {}, {}, {}, {}, {}".
format(*numbers))
```

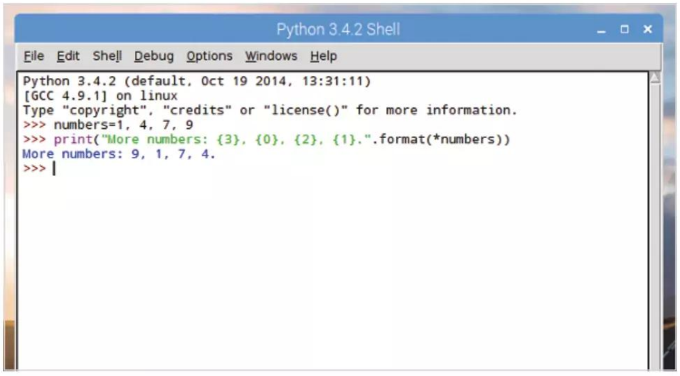

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> numbers=1, 3, 45, 567546, 3425346345
>>> print("Some numbers: {}, {}, {}, {}, {}".format(*numbers))
Some numbers: 1, 3, 45, 567546, 3425346345
>>>
```

STEP 6

With indexing in lists, the same applies to calling a list using string formatting. You can index each item according to its position (from 0 to however many are present):

```
numbers=1, 4, 7, 9
print("More numbers: {3}, {0}, {2},
{1}.".format(*numbers))
```

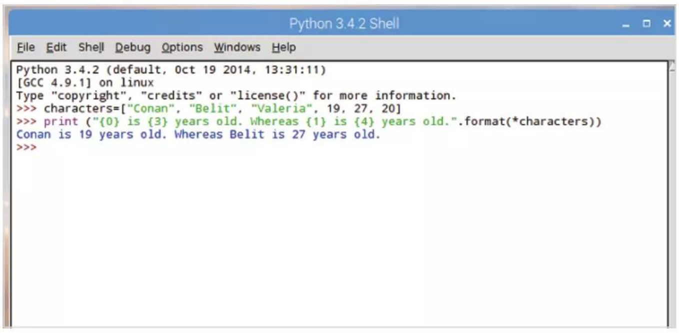

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> numbers=1, 4, 7, 9
>>> print("More numbers: {3}, {0}, {2}, {1}.".format(*numbers))
More numbers: 9, 1, 7, 4.
>>> |
```

STEP 7

And as you probably suspect, you can mix strings and integers in a single list to be called in the .format function:

```
characters=["Conan", "Belit", "Valeria", 19, 27,
20]
print ("{0} is {3} years old. Whereas {1} is {4}
years old.".format(*characters))
```

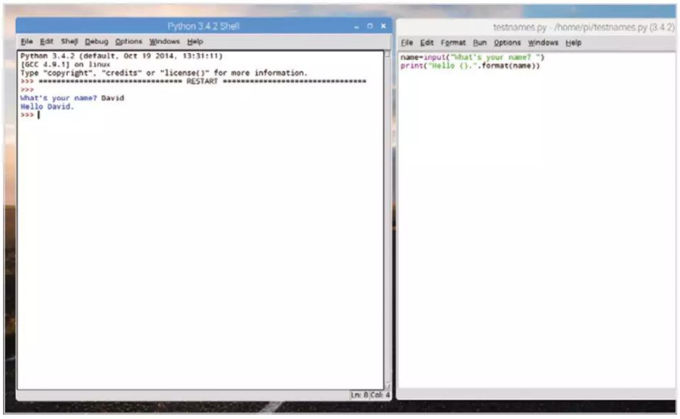

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> characters=["Conan", "Belit", "Valeria", 19, 27, 20]
>>> print ("{0} is {3} years old. Whereas {1} is {4} years old.".format(*characters))
Conan is 19 years old. Whereas Belit is 27 years old.
>>>
```

STEP 8

You can also print out the content of a user's input in the same fashion:

```
name=input("What's your name? ")
print("Hello {}.".format(name)
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
testnames.py - /home/pi/testnames.py (3.4.2)
File Edit Format Run Options Windows Help
Ln: 8 Col: 4

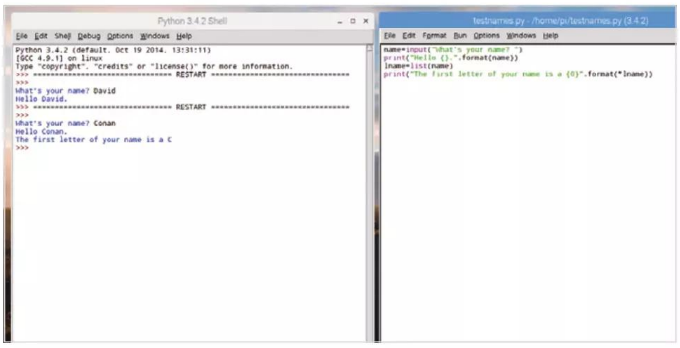
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
What's your name? David
Hello David.
>>> |
```

```
name=input("What's your name? ")
print("Hello {}.".format(name))
```

STEP 9

You can extend this simple code example to display the first letter in a person's entered name:

```
name=input("What's your name? ")
print("Hello {}.".format(name))
lname=list(name)
print("The first letter of your name is a {0}".
format(*lname))
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
testnames.py - /home/pi/testnames.py (3.4.2)
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
What's your name? David
Hello David.
>>> ================================ RESTART ================================
>>>
What's your name? Conan
Hello Conan.
The first letter of your name is a C
>>>
```

```
name=input("What's your name? ")
print("Hello {}.".format(name))
lname=list(name)
print("The first letter of your name is a {0}".format(*lname))
```

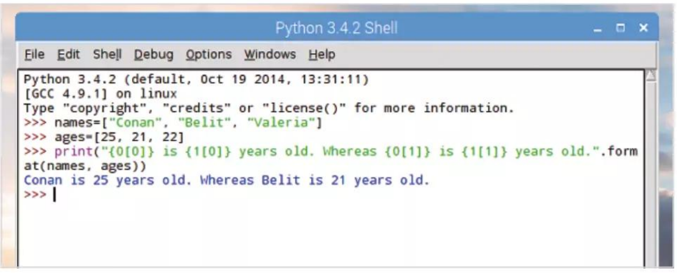
STEP 10

You can also call upon a pair of lists and reference them individually within the same print function. Looking back the code from Step 7, you can alter it with:

```
names=["Conan", "Belit", "Valeria"]
ages=[25, 21, 22]
```

Creating two lists. Now you can call each list, and individual items:

```
print("{0[0]} is {1[0]} years old. Whereas {0[1]}
is {1[1]} years old.".format(names, ages))
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> names=["Conan", "Belit", "Valeria"]
>>> ages=[25, 21, 22]
>>> print("{0[0]} is {1[0]} years old. Whereas {0[1]} is {1[1]} years old.".form
at(names, ages))
Conan is 25 years old. Whereas Belit is 21 years old.
>>> |
```

Date and Time

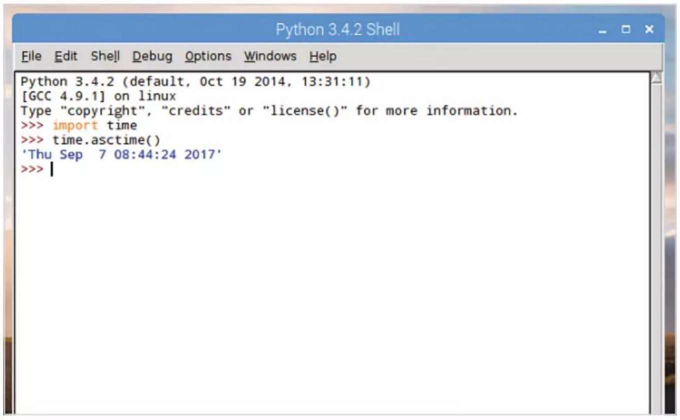
When working with data it's often handy to have access to the time. For example, you may want to time-stamp an entry or see at what time a user logged into the system and for how long. Luckily acquiring the date and time is easy, thanks to the Time module.

TIME LORDS

The time module contains functions that help you retrieve the current system time, reads the date from strings, formats the time and date and much more.

STEP 1 First you need to import the time module. It's one that's built-in to Python 3 so you shouldn't need to drop into a command prompt and pip install it. Once it's imported, you can call the current time and date with a simple command:

```
import time
time.asctime()
```



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import time
>>> time.asctime()
'Thu Sep  7 08:44:24 2017'
>>> 
```

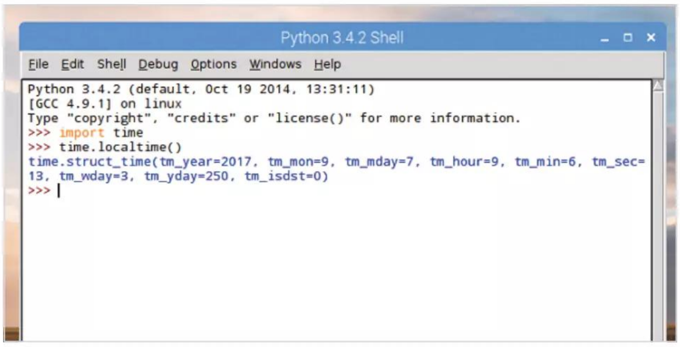
STEP 2 The time function is split into nine tuples, these are divided up into indexed items, as with any other tuple, and shown in the screen shot below.

Index	Field	Values
0	4-digit year	2016
1	Month	1 to 12
2	Day	1 to 31
3	Hour	0 to 23
4	Minute	0 to 59
5	Second	0 to 61 (60 or 61 are leap-seconds)
6	Day of Week	0 to 6 (0 is Monday)
7	Day of year	1 to 366 (Julian day)
8	Daylight savings	-1, 0, 1, -1 means library determines DST

STEP 3 You can see the structure of how time is presented by entering:

```
time.local.time()
```

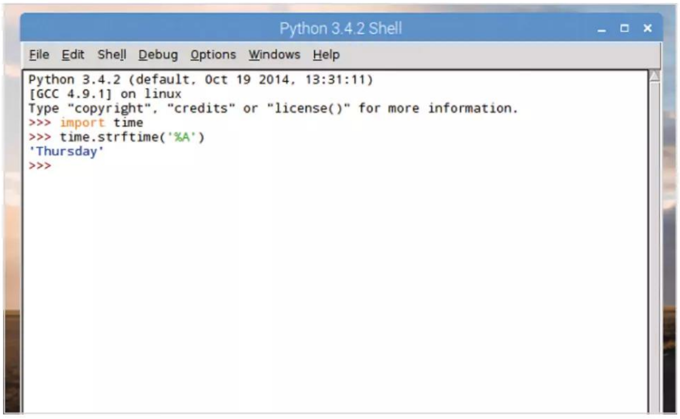
The output is displayed as such: `'time.struct_time(tm_year=2017, tm_mon=9, tm_mday=7, tm_hour=9, tm_min=6, tm_sec=13, tm_wday=3, tm_yday=250, tm_isdst=0)'`; obviously dependent on your current time as opposed to the time shown above.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import time
>>> time.localtime()
time.struct_time(tm_year=2017, tm_mon=9, tm_mday=7, tm_hour=9, tm_min=6, tm_sec=
13, tm_wday=3, tm_yday=250, tm_isdst=0)
>>> 
```

STEP 4 There are numerous functions built into the time module. One of the most common of these is .strftime(). With it, you're able to present a wide range of arguments as it converts the time tuple into a string. For example, to display the current day of the week you can use:

```
time.strftime('%A')
```

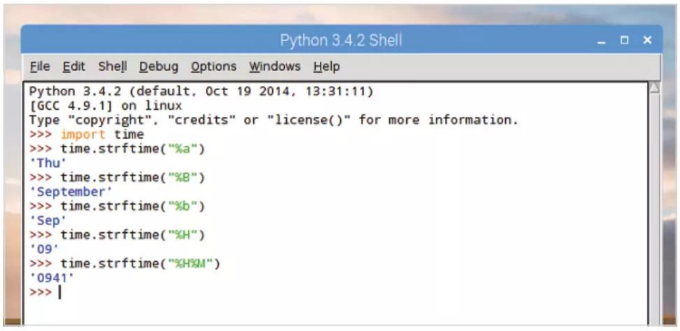


```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import time
>>> time.strftime('%A')
'Thursday'
>>> 
```

STEP 5

This naturally means you can incorporate various functions into your own code, such as:

```
time.strftime("%a")
time.strftime("%B")
time.strftime("%b")
time.strftime("%H")
time.strftime("%H%M")
```

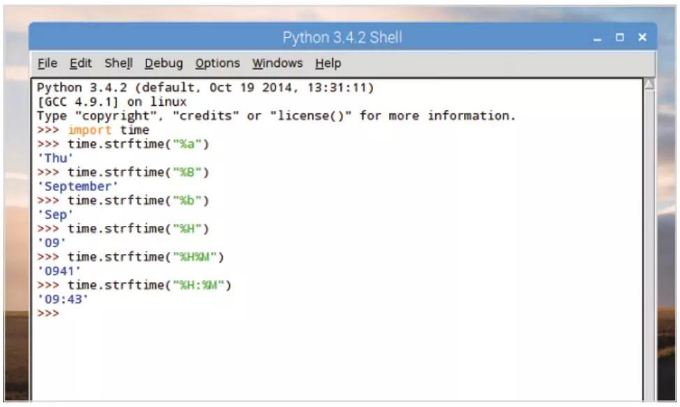

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import time
>>> time.strftime("%a")
'Thu'
>>> time.strftime("%B")
'September'
>>> time.strftime("%b")
'Sep'
>>> time.strftime("%H")
'09'
>>> time.strftime("%H%M")
'0941'
>>> |
```

STEP 6

Note the last two entries, with %H and %H%M, as you can see these are the hours and minutes and as the last entry indicates, entering them as %H%M doesn't display the time correctly in the Shell. You can easily rectify this with:

```
time.strftime("%H:%M")
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

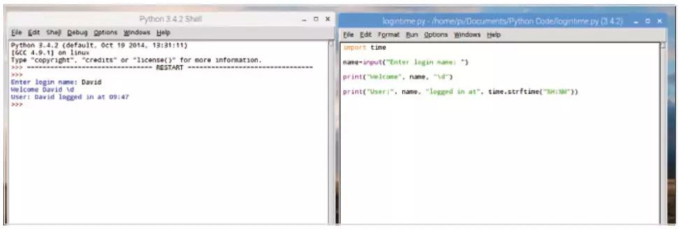
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import time
>>> time.strftime("%a")
'Thu'
>>> time.strftime("%B")
'September'
>>> time.strftime("%b")
'Sep'
>>> time.strftime("%H")
'09'
>>> time.strftime("%H%M")
'0941'
>>> time.strftime("%H:%M")
'09:43'
>>>
```

STEP 7

This means you're going to be able to display either the current time or the time when something occurred, such as a user entering their name. Try this code in the Editor:

```
import time
name=input("Enter login name: ")
print("Welcome", name, "\d")
print("User:", name, "logged in at", time.
strftime("%H:%M"))
```

Try to extend it further to include day, month, year and so on.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
logintime.py - /home/pi/Documents/Python Code/logintime.py (3.4.2)
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Enter login name: David
Welcome David \d
User: David logged in at 09:47
>>>
```

```
import time

name=input("Enter login name: ")

print("Welcome", name, "\d")

print("User:", name, "logged in at", time.strftime("%H:%M"))
```

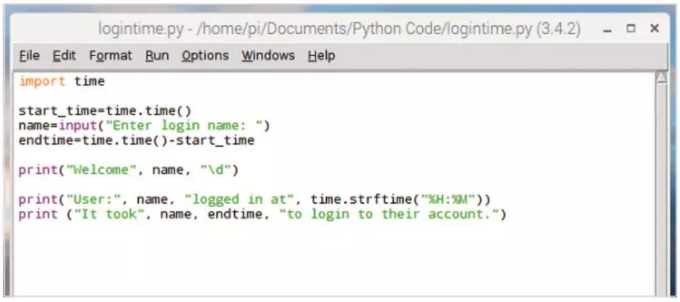
STEP 8

You saw at the end of the previous section, in the code to calculate Pi to however many decimal places the users wanted, you can time a particular event in Python. Take the code from above and alter it slightly by including:

```
start_time=time.time()
```

Then there's:

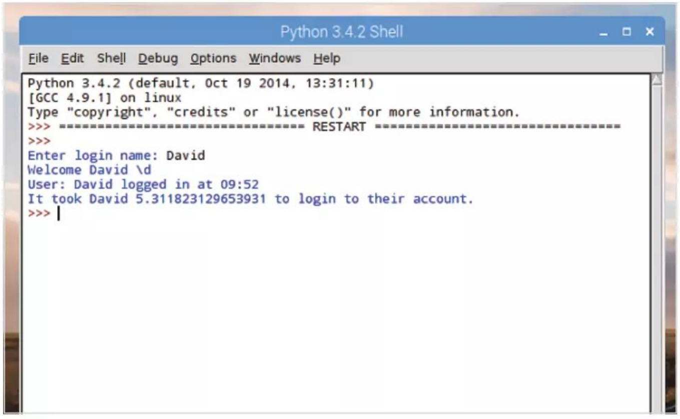
```
endtime=time.time()-start_time
```


logintime.py - /home/pi/Documents/Python Code/logintime.py (3.4.2)
File Edit Format Run Options Windows Help

```
import time

start_time=time.time()
name=input("Enter login name: ")
endtime=time.time()-start_time

print("Welcome", name, "\d")

print("User:", name, "logged in at", time.strftime("%H:%M"))
print ("It took", name, endtime, "to login to their account.")
```

STEP 9

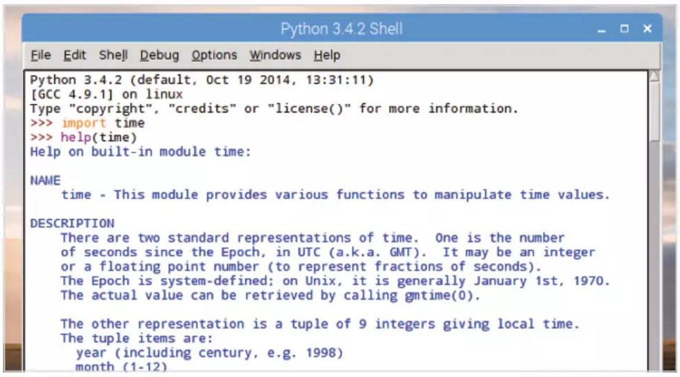
The output will look similar to the screenshot below. The timer function needs to be either side of the input statement, as that's when the variable name is being created, depending on how long the user took to log in. The length of time is then displayed on the last line of the code as the `endtime` variable.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Enter login name: David
Welcome David \d
User: David logged in at 09:52
It took David 5.311823129653931 to login to their account.
>>> |
```

STEP 10

There's a lot that can be done with the time module; some of it is quite complex too, such as displaying the number of seconds since January 1st 1970. If you want to drill down further into the time module, then in the Shell enter: `help(time)` to display the current Python version help file for the time module.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import time
>>> help(time)
Help on built-in module time:

NAME
    time - This module provides various functions to manipulate time values.

DESCRIPTION
    There are two standard representations of time.  One is the number
    of seconds since the Epoch, in UTC (a.k.a. GMT).  It may be an integer
    or a floating point number (to represent fractions of seconds).
    The Epoch is system-defined; on Unix, it is generally January 1st, 1970.
    The actual value can be retrieved by calling gmtime(0).

    The other representation is a tuple of 9 integers giving local time.
    The tuple items are:
      year (including century, e.g. 1998)
      month (1-12)
```

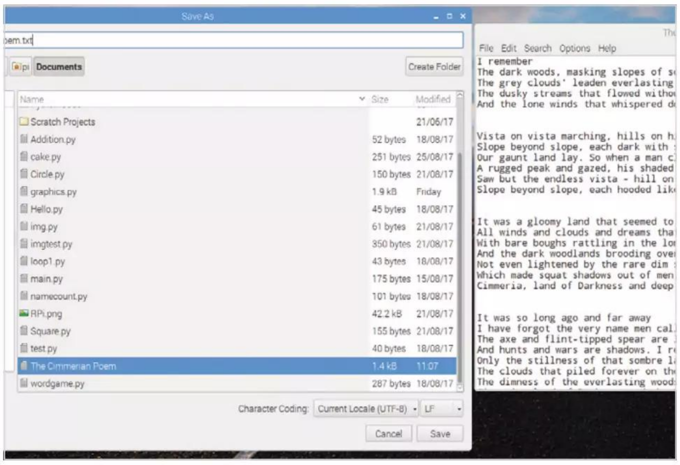
Opening Files

In Python you can read text and binary files in your programs. You can also write to file, which is something we will look at next. Reading and writing to files enables you to output and store data from your programs.

OPEN, READ AND WRITE

In Python you create a file object, similar to creating a variable, only pass in the file using the open() function. Files are usually categorised as text or binary.

STEP 1 Start by entering some text into your system's text editor. The text editor is best, not a word processor, as word processors include background formatting and other elements. In our example, we have the poem The Cimmerian, by Robert E Howard. You need to save the file as poem.txt.



oem.txt

pi Documents

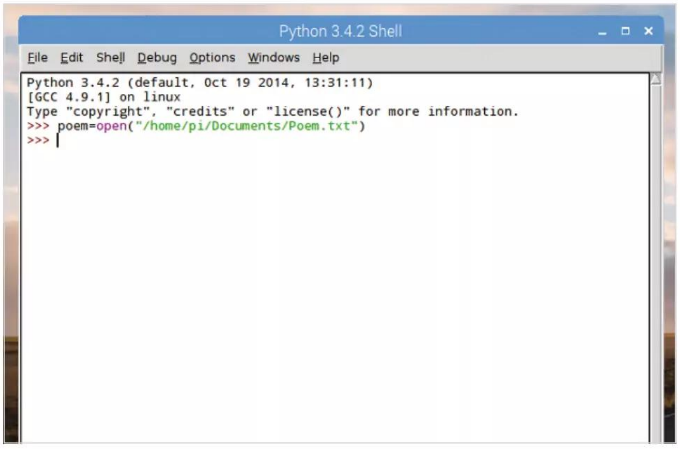
Name	Size	Modified
Scratch Projects		21/06/17
Addition.py	52 bytes	18/08/17
cake.py	251 bytes	25/08/17
Circle.py	150 bytes	21/08/17
graphics.py	1.9 kB	Friday
Hello.py	45 bytes	18/08/17
img.py	61 bytes	21/08/17
imgtest.py	350 bytes	21/08/17
loop1.py	43 bytes	18/08/17
main.py	175 bytes	15/08/17
namecount.py	101 bytes	18/08/17
RPi.png	42.2 kB	21/08/17
Square.py	155 bytes	21/08/17
test.py	40 bytes	18/08/17
The Cimmerian Poem	1.4 kB	11:07
wordgame.py	287 bytes	18/08/17

Character Coding: Current Locale (UTF-8) LF

```
I remember
The dark woods, masking slopes of s
The grey clouds' leaden everlasting
The dusky streams that flowed witho
And the lone winds that whispered d


Vista on vista marching, hills on h
Slope beyond slope, each dark with
Our gaunt land lay. So when a man c
A rugged peak and gazed, his shaded
Saw but the endless vista - hill on
Slope beyond slope, each hooded lik


It was a gloomy land that seemed to
All winds and clouds and dreams tha
With bare boughs rattling in the lo
And the dark woodlands brooding ove
Not even lightened by the rare dim
Which made squat shadows out of men
Cimmeria, land of Darkness and deep


It was so long ago and far away
I have forgot the very name men cal
The axe and flint-tipped spear are
And hunts and wars are shadows. I r
Only the stillness of that sombre l
The clouds that piled forever on th
The dimness of the everlasting wood
```

STEP 2 You use the open() function to pass the file into a variable as an object. You can name the file object anything you like, but you will need to tell Python the name and location of the text file you're opening:

```
poem=open("/home/pi/Documents/Poem.txt")
```

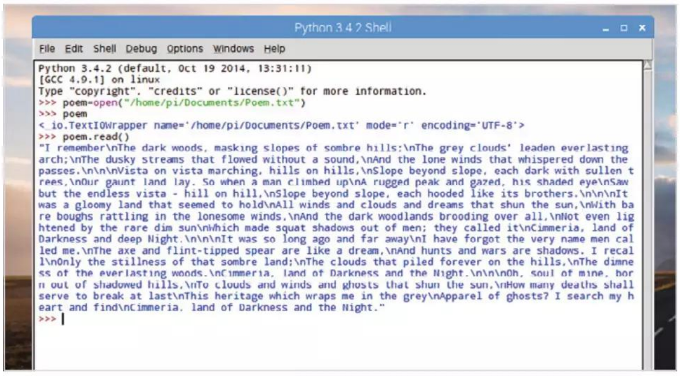


```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> poem=open("/home/pi/Documents/Poem.txt")
>>> 
```

STEP 3 If you now enter poem into the Shell, you will get some information regarding the text file you've just asked to be opened. You can now use the poem variable to read the contents of the file:

```
poem.read()
```

Note than a /n entry in the text represents a new line, as you used previously.

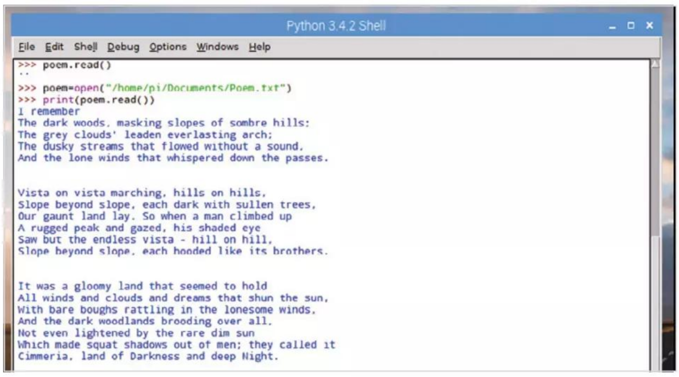


```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> poem=open("/home/pi/Documents/Poem.txt")
>>> poem
<_io.TextIOWrapper name='/home/pi/Documents/Poem.txt' mode='r' encoding='UTF-8'>
>>> poem.read()
"I remember\nThe dark woods, masking slopes of sombre hills:\nThe grey clouds' leaden everlasting
arch;\nThe dusky streams that flowed without a sound,\nAnd the lone winds that whispered down the
passes.\n\n\nVista on vista marching, hills on hills,\nSlope beyond slope, each dark with sullen t
rees,\nOur gaunt land lay. So when a man climbed up\nA rugged peak and gazed, his shaded eye\nSaw
but the endless vista - hill on hill,\nSlope beyond slope, each hooded like its brothers.\n\n\nIt
was a gloomy land that seemed to hold\nAll winds and clouds and dreams that shun the sun,\nWith ba
re boughs rattling in the lonesome winds,\nAnd the dark woodlands brooding over all,\nNot even lig
htened by the rare dim sun\nWhich made squat shadows out of men; they called it\nCimmeria, land of
Darkness and deep Night.\n\n\nIt was so long ago and far away\nI have forgot the very name men cal
led me.\nThe axe and flint-tipped spear are like a dream,\nAnd hunts and wars are shadows. I recal
l\nOnly the stillness of that sombre land;\nThe clouds that piled forever on the hills,\nThe dimne
ss of the everlasting woods.\nCimmeria, land of Darkness and the Night.\n\n\nOh, soul of mine, bor
n out of shadowed hills,\nTo clouds and winds and ghosts that shun the sun,\nHow many deaths shall
serve to break at last\nThis heritage which wraps me in the grey\nApparel of ghosts? I search my h
eart and find\nCimmeria, land of Darkness and the Night."
>>> |
```

STEP 4 If you enter poem.read() a second time you will notice that the text has been removed from the file. You will need to enter: `poem=open("/home/pi/Documents/Poem.txt")` again to recreate the file. This time, however, enter:

```
print(poem.read())
```

This time, the /n entries are removed in favour of new lines and readable text.



```
>>> poem.read()
''
>>> poem=open("/home/pi/Documents/Poem.txt")
>>> print(poem.read())
I remember
The dark woods, masking slopes of sombre hills:
The grey clouds' leaden everlasting arch;
The dusky streams that flowed without a sound,
And the lone winds that whispered down the passes.


Vista on vista marching, hills on hills,
Slope beyond slope, each dark with sullen trees,
Our gaunt land lay. So when a man climbed up
A rugged peak and gazed, his shaded eye
Saw but the endless vista - hill on hill,
Slope beyond slope, each hooded like its brothers.


It was a gloomy land that seemed to hold
All winds and clouds and dreams that shun the sun,
With bare boughs rattling in the lonesome winds,
And the dark woodlands brooding over all,
Not even lightened by the rare dim sun
Which made squat shadows out of men; they called it
Cimmeria, land of Darkness and deep Night.
```

STEP 5

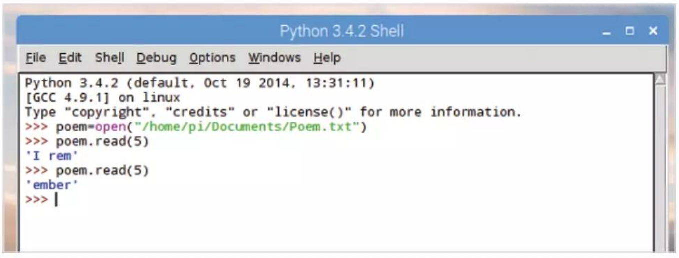
Just as with lists, tuples, dictionaries and so on, you're able to index individual characters of the text. For example:

```
poem.read(5)
```

Displays the first five characters, whilst again entering:

```
poem.read(5)
```

Will display the next five. Entering (1) will display one character at a time.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> poem=open("/home/pi/Documents/Poem.txt")
>>> poem.read(5)
'I rem'
>>> poem.read(5)
'ember'
>>> |
```

STEP 6

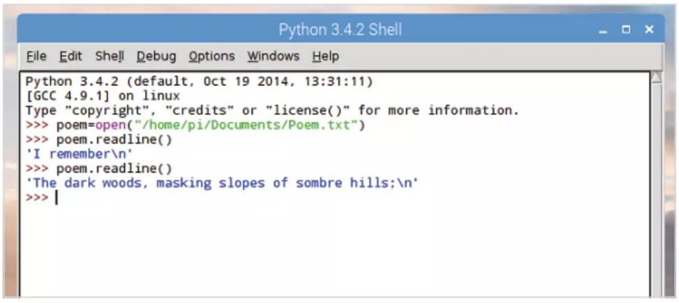
Similarly, you can display one line of text at a time by using the readline() function. For example:

```
poem=open("/home/pi/Documents/Poem.txt")
poem.readline()
```

Will display the first line of the text with:

```
poem.readline()
```

Displaying the next line of text once more.

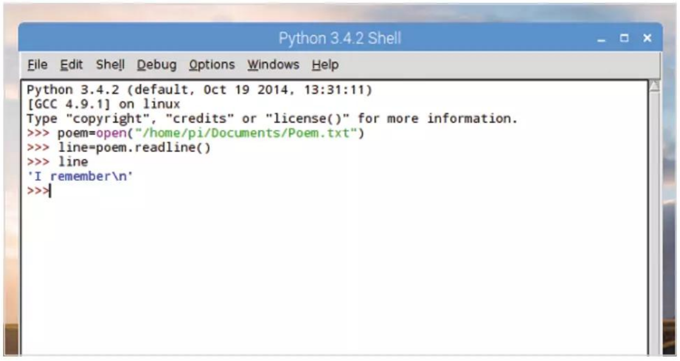

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> poem=open("/home/pi/Documents/Poem.txt")
>>> poem.readline()
'I remember\n'
>>> poem.readline()
'The dark woods, masking slopes of sombre hills;\n'
>>> |
```

STEP 7

You may have guessed that you can pass the readline() function into a variable, thus allowing you to call it again when needed:

```
poem=open("/home/pi/Documents/Poem.txt")
line=poem.readline()
line
```

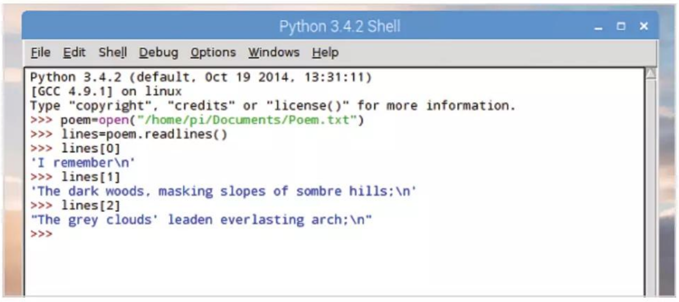

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> poem=open("/home/pi/Documents/Poem.txt")
>>> line=poem.readline()
>>> line
'I remember\n'
>>>|
```

STEP 8

Extending this further, you can use readlines() to grab all the lines of the text and store them as multiple lists. These can then be stored as a variable:

```
poem=open("/home/pi/Documents/Poem.txt")
lines=poem.readlines()
lines[0]
lines[1]
lines[2]
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> poem=open("/home/pi/Documents/Poem.txt")
>>> lines=poem.readlines()
>>> lines[0]
'I remember\n'
>>> lines[1]
'The dark woods, masking slopes of sombre hills;\n'
>>> lines[2]
"The grey clouds' leaden everlasting arch;\n"
>>>
```

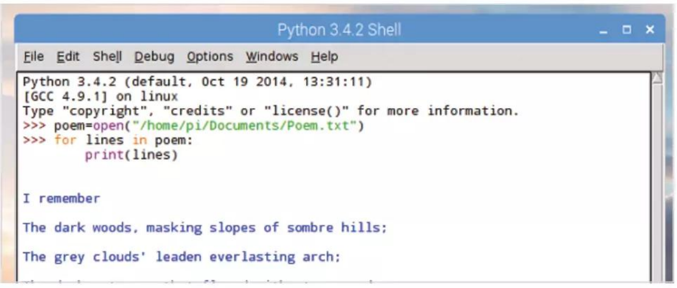
STEP 9

You can also use the for statement to read the lines of text back to us:

```
for lines in lines:
   print(lines)
```

Since this is Python, there are other ways to produce the same output:

```
poem=open("/home/pi/Documents/Poem.txt")
for lines in poem:
   print(lines)
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

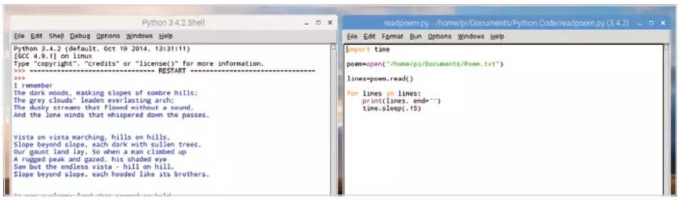
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> poem=open("/home/pi/Documents/Poem.txt")
>>> for lines in poem:
        print(lines)


I remember

The dark woods, masking slopes of sombre hills;

The grey clouds' leaden everlasting arch;
```

STEP 10

Let's imagine that you want to print the text one character at a time, like an old dot matrix printer would. You can use the time module mixed with what you've looked at here. Try this:

```
import time
poem=open("/home/pi/Documents/Poem.txt")
lines=poem.read()
for lines in lines:
  print(lines, end="")
  time.sleep(.15)
```

The output is fun to view, and easily incorporated into your own code.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
readpoem.py - /home/pi/Documents/Python Code/readpoem.py (3.4.2)
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
I remember
The dark woods, masking slopes of sombre hills;
The grey clouds' leaden everlasting arch;
The dusky streams that flowed without a sound,
And the lone winds that whispered down the passes.


Vista on vista marching, hills on hills,
Slope beyond slope, each dark with sullen trees,
Our gaunt land lay. So when a man climbed up
A rugged peak and gazed, his shaded eye
Saw but the endless vista - hill on hill,
Slope beyond slope, each hooded like its brothers.
```

```
import time

poem=open("/home/pi/Documents/Poem.txt")

lines=poem.read()

for lines in lines:
    print(lines, end="")
    time.sleep(.15)
```

Writing to Files

The ability to read external files within Python is certainly handy but writing to a file is better still. Using the write() function, you're able to output the results of a program to a file, that you can then read() back into Python.

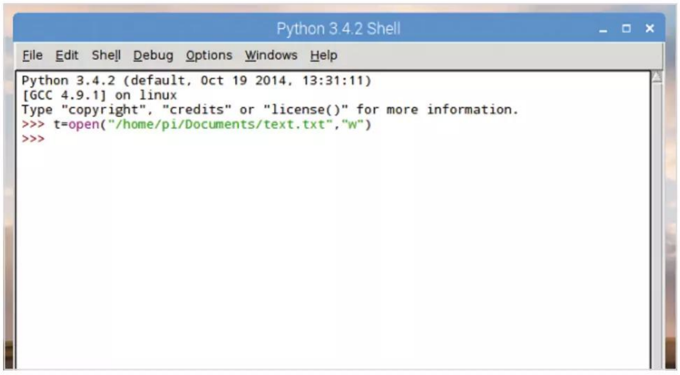
WRITE AND CLOSE

The write() function is slightly more complex than read(). Along with the filename you must also include an access mode which determines whether the file in question is in read or write mode.

STEP 1 Start by opening IDLE and enter the following:

```
t=open("/home/pi/Documents/text.txt","w")
```

Change the destination from /home/pi/Documents to your own system. This code will create a text file called text.txt in write mode using the variable 't'. If there's no file of that name in the location, it will create one. If one already exits, it will overwrite it, so be careful.

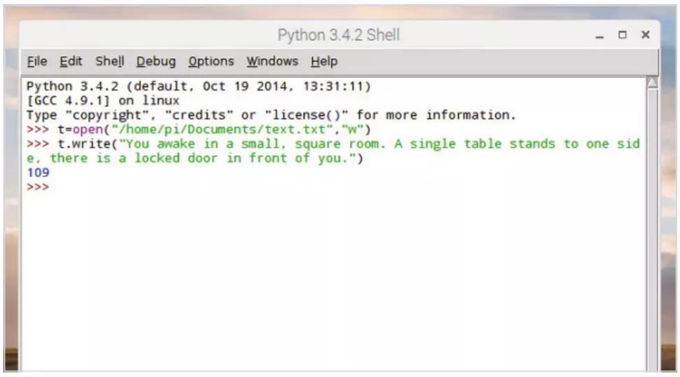


```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> t=open("/home/pi/Documents/text.txt","w")
>>>
```

STEP 2 You can now write to the text file using the write() function. This works opposite to read(), writing lines instead of reading them. Try this:

```
t.write("You awake in a small, square room. A single table stands to one side, there is a locked door in front of you.")
```

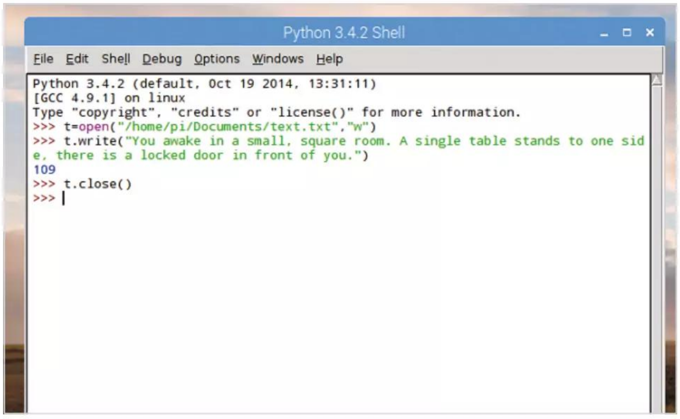
Note the 109. It's the number of characters you've entered.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> t=open("/home/pi/Documents/text.txt","w")
>>> t.write("You awake in a small, square room. A single table stands to one sid
e, there is a locked door in front of you.")
109
>>>
```

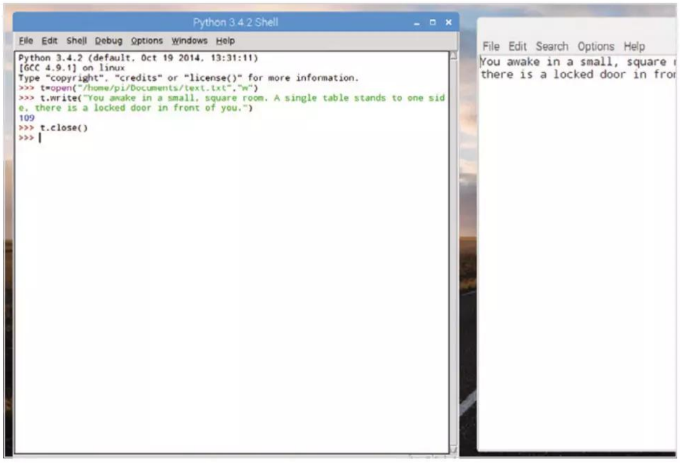
STEP 3 However, the actual text file is still blank (you can check by opening it up). This is because you've written the line of text to the file object but not committed it to the file itself. Part of the write() function is that you need to commit the changes to the file; you can do this by entering:

```
t.close()
```



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> t=open("/home/pi/Documents/text.txt","w")
>>> t.write("You awake in a small, square room. A single table stands to one sid
e, there is a locked door in front of you.")
109
>>> t.close()
>>> |
```

STEP 4 If you now open the text file with a text editor, you can see that the line you created has been written to the file. This gives us the foundation for some interesting possibilities: perhaps the creation of your own log file or even the beginning of an adventure game.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
File Edit Search Options Help

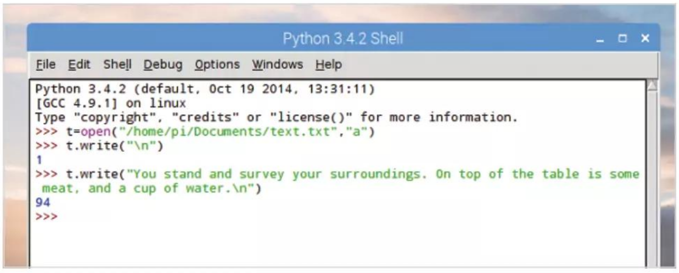
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> t=open("/home/pi/Documents/text.txt","w")
>>> t.write("You awake in a small, square room. A single table stands to one sid
e, there is a locked door in front of you.")
109
>>> t.close()
>>> |
```

```
You awake in a small, square r
there is a locked door in fron
```

STEP 5

To expand this code, you can reopen the file using 'a', for access or append mode. This will add any text at the end of the original line instead of wiping the file and creating a new one. For example:

```
t=open("/home/pi/Documents/text.txt","a")
t.write("\n")
t.write(" You stand and survey your surroundings.
On top of the table is some meat, and a cup of
water.\n")
```

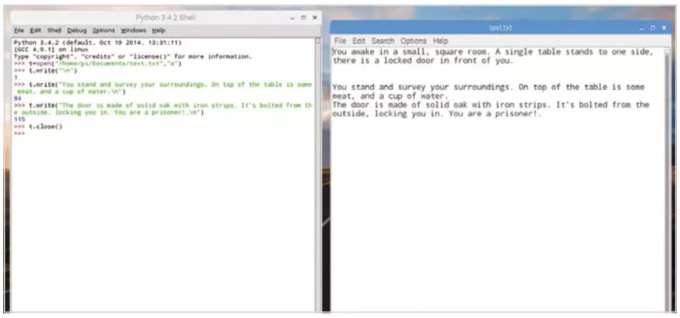

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> t=open("/home/pi/Documents/text.txt","a")
>>> t.write("\n")
1
>>> t.write("You stand and survey your surroundings. On top of the table is some
 meat, and a cup of water.\n")
94
>>>
```

STEP 6

You can keep extending the text line by line, ending each with a new line (\n). When you're done, finish the code with t.close() and open the file in a text editor to see the results:

```
t.write("The door is made of solid oak with iron
strips. It's bolted from the outside, locking you
in. You are a prisoner!.\n")
t.close()
```

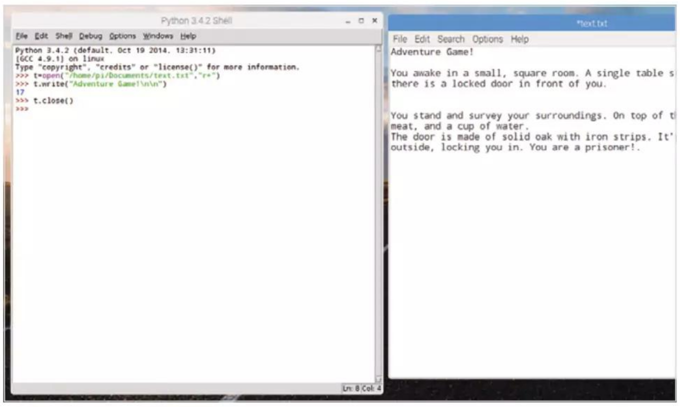

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
text.txt
File Edit Search Options Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> t=open("/home/pi/Documents/text.txt","a")
>>> t.write("\n")
1
>>> t.write("You stand and survey your surroundings. On top of the table is some
 meat, and a cup of water.\n")
94
>>> t.write("The door is made of solid oak with iron strips. It's bolted from th
e outside, locking you in. You are a prisoner!.\n")
115
>>> t.close()
>>>
```

```
You awake in a small, square room. A single table stands to one side,
there is a locked door in front of you.


You stand and survey your surroundings. On top of the table is some
meat, and a cup of water.
The door is made of solid oak with iron strips. It's bolted from the
outside, locking you in. You are a prisoner!.
```

STEP 7

There are various types of file access to consider using the open() function. Each depends on how the file is accessed and even the position of the cursor. For example, r+ opens a file in read and write and places the cursor at the start of the file.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
*text.txt
File Edit Search Options Help
Ln: 8 Col: 4

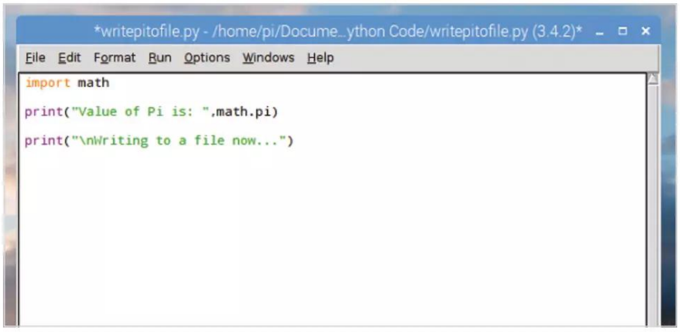
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> t=open("/home/pi/Documents/text.txt","r+")
>>> t.write("Adventure Game!\n\n")
17
>>> t.close()
>>>
```

```
Adventure Game!

You awake in a small, square room. A single table s
there is a locked door in front of you.


You stand and survey your surroundings. On top of t
meat, and a cup of water.
The door is made of solid oak with iron strips. It'
outside, locking you in. You are a prisoner!.
```

STEP 8

You can pass variables to a file that you've created in Python. Perhaps you want the value of Pi to be written to a file. You can call Pi from the math module, create a new file and pass the output of Pi into the new file:

```
import math
print("Value of Pi is: ",math.pi)
print("\nWriting to a file now…")
```


writepitofile.py - /home/pi/Docume...ython Code/writepitofile.py (3.4.2)
File Edit Format Run Options Windows Help

```
import math

print("Value of Pi is: ",math.pi)

print("\nWriting to a file now...")
```

STEP 9

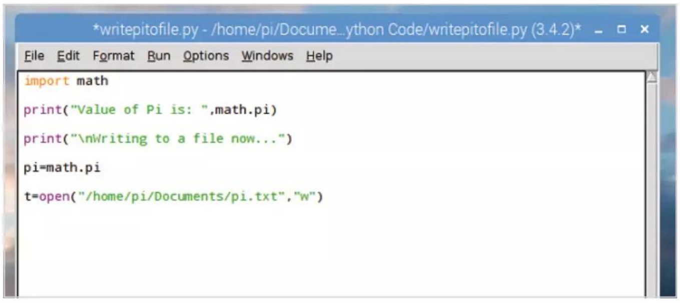
Now let's create a variable called pi and assign it the value of Pi:

```
pi=math.pi
```

You also need to create a new file in which to write Pi to:

```
t=open("/home/pi/Documents/pi.txt","w")
```

Remember to change your file location to your own particular system setup.


writepitofile.py - /home/pi/Docume...ython Code/writepitofile.py (3.4.2)
File Edit Format Run Options Windows Help

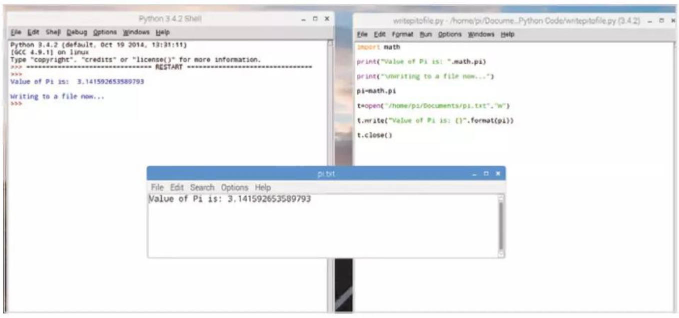
```
import math

print("Value of Pi is: ",math.pi)

print("\nWriting to a file now...")

pi=math.pi

t=open("/home/pi/Documents/pi.txt","w")
```

STEP 10

To finish, you can use string formatting to call the variable and write it to the file, then commit the changes and close the file:

```
t.write("Value of Pi is: {}".format(pi))
t.close()
```

You can see from the results that you're able to pass any variable to a file.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
writepitofile.py - /home/pi/Docume...Python Code/writepitofile.py (3.4.2)
File Edit Format Run Options Windows Help
pi.txt
File Edit Search Options Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Value of Pi is:  3.141592653589793

Writing to a file now...
>>>
```

```
import math

print("Value of Pi is: ",math.pi)

print("\nWriting to a file now...")

pi=math.pi

t=open("/home/pi/Documents/pi.txt","w")

t.write("Value of Pi is: {}".format(pi))

t.close()
```

```
Value of Pi is: 3.141592653589793
```

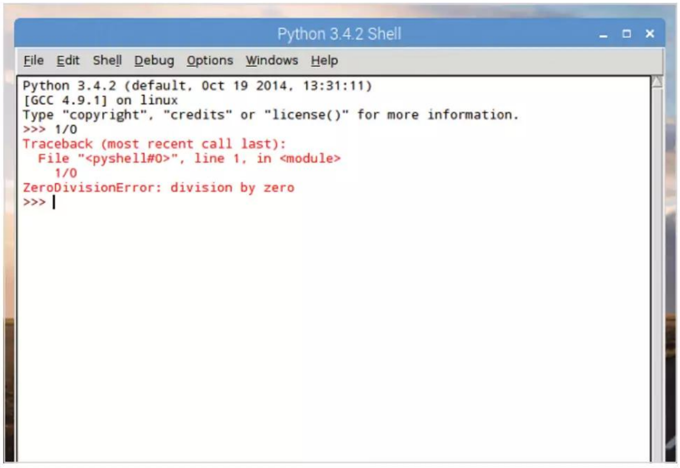
Exceptions

When coding, you'll naturally come across some issues that are out of your control. Let's assume you ask a user to divide two numbers and they try to divide by zero. This will create an error and break your code.

EXCEPTIONAL OBJECTS

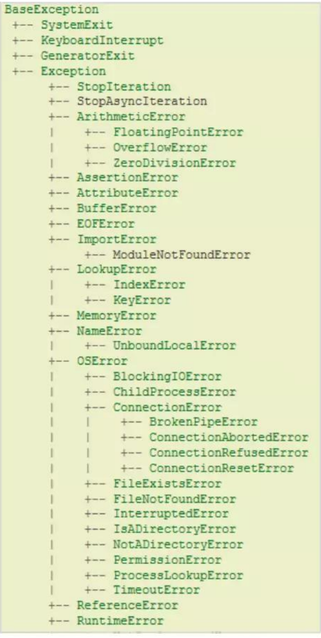
Rather than stop the flow of your code, Python includes exception objects which handle unexpected errors in the code. You can combat errors by creating conditions where exceptions may occur.

STEP 1 You can create an exception error by simply trying to divide a number by zero. This will report back with the ZeroDivisionError: Division by zero message, as seen in the screenshot. The ZeroDivisionError part is the exception class, of which there are many.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> 1/0
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    1/0
ZeroDivisionError: division by zero
>>> 
```

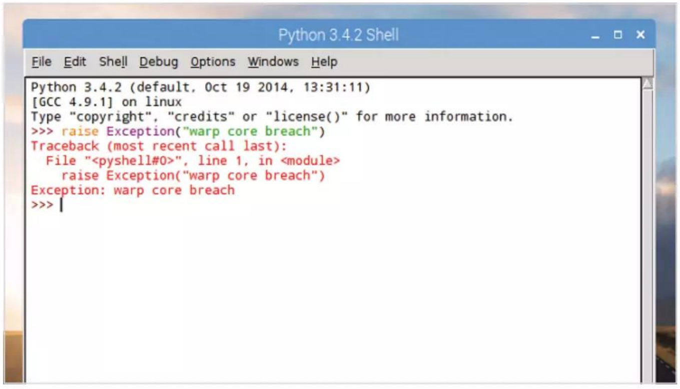
STEP 2 Most exceptions are raised automatically when Python comes across something that's inherently wrong with the code. However, you can create your own exceptions that are designed to contain the potential error and react to it, as opposed to letting the code fail.



```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StopAsyncIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- AssertionError
      +-- AttributeError
      +-- BufferError
      +-- EOFError
      +-- ImportError
           +-- ModuleNotFoundError
      +-- LookupError
      |    +-- IndexError
      |    +-- KeyError
      +-- MemoryError
      +-- NameError
      |    +-- UnboundLocalError
      +-- OSError
      |    +-- BlockingIOError
      |    +-- ChildProcessError
      |    +-- ConnectionError
      |    |    +-- BrokenPipeError
      |    |    +-- ConnectionAbortedError
      |    |    +-- ConnectionRefusedError
      |    |    +-- ConnectionResetError
      |    +-- FileExistsError
      |    +-- FileNotFoundError
      |    +-- InterruptedError
      |    +-- IsADirectoryError
      |    +-- NotADirectoryError
      |    +-- PermissionError
      |    +-- ProcessLookupError
      |    +-- TimeoutError
      +-- ReferenceError
      +-- RuntimeError
```

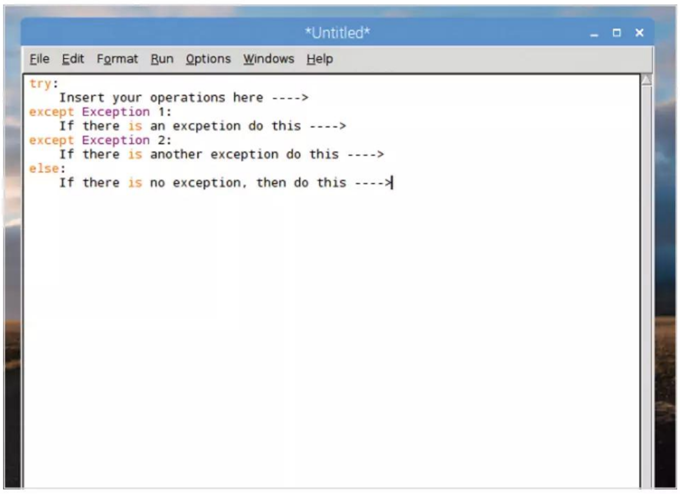
STEP 3 You can use the functions raise exception to create our own error handling code within Python. Let's assume your code has you warping around the cosmos, too much however results in a warp core breach. To stop the game from exiting due to the warp core going supernova, you can create a custom exception:

`raise Exception("warp core breach")`



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> raise Exception("warp core breach")
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    raise Exception("warp core breach")
Exception: warp core breach
>>> 
```

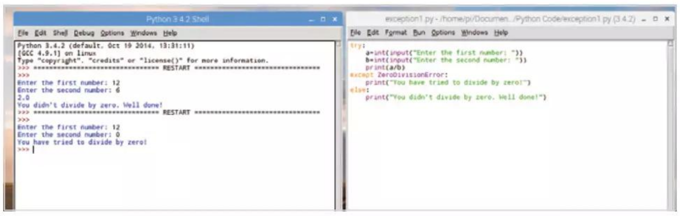
STEP 4 To trap any errors in the code you can encase the potential error within a try: block. This block consists of try, except, else, where the code is held within try:, then if there's an exception do something, else do something else.



```
try:
    Insert your operations here ---->
except Exception 1:
    If there is an excpetion do this ---->
except Exception 2:
    If there is another exception do this ---->
else:
    If there is no exception, then do this ---->
```

STEP 5 For example, use the divide by zero error. You can create an exception where the code can handle the error without Python quitting due to the problem:

```
try:
  a=int(input("Enter the first number: "))
  b=int(input("Enter the second number: "))
  print(a/b)
except ZeroDivisionError:
  print("You have tried to divide by zero!")
else:
  print("You didn't divide by zero. Well done!")
```

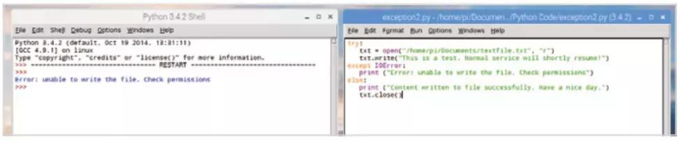


```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>> 
Enter the first number: 12
Enter the second number: 6
2.0
You didn't divide by zero. Well done!
>>> ================================ RESTART ================================
>>> 
Enter the first number: 12
Enter the second number: 0
You have tried to divide by zero!
>>> 
```

```
try:
    a=int(input("Enter the first number: "))
    b=int(input("Enter the second number: "))
    print(a/b)
except ZeroDivisionError:
    print("You have tried to divide by zero!")
else:
    print("You didn't divide by zero. Well done!")
```

STEP 6 You can use exceptions to handle a variety of useful tasks. Using an example from our previous tutorials, let's assume you want to open a file and write to it:

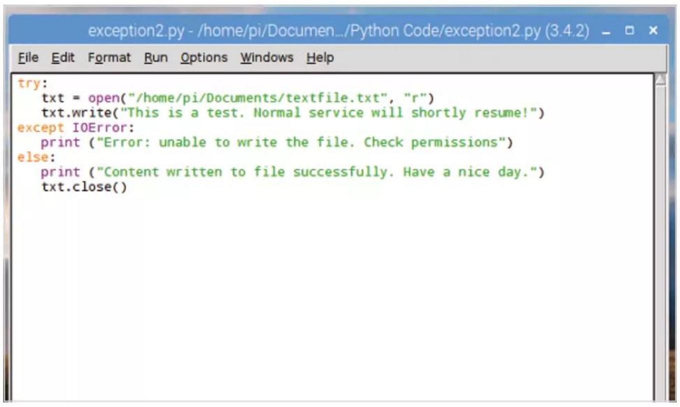
```
try:
 txt = open("/home/pi/Documents/textfile.txt", "r")
 txt.write("This is a test. Normal service will 
shortly resume!")
except IOError:
 print ("Error: unable to write the file. Check 
permissions")
else:
 print ("Content written to file successfully. Have 
a nice day.")
 txt.close()
```



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>> 
Error: unable to write the file. Check permissions
>>> 
```

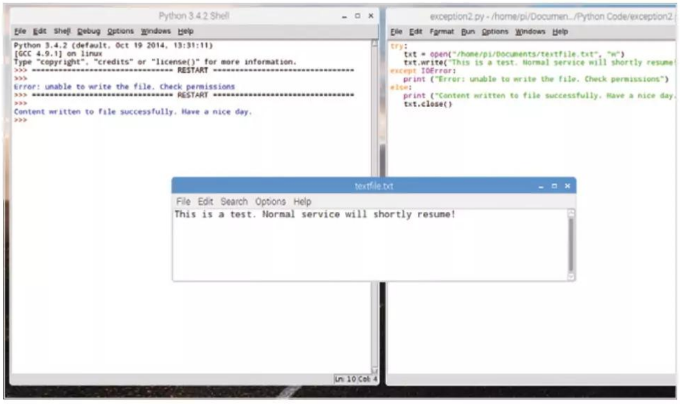
```
try:
    txt = open("/home/pi/Documents/textfile.txt", "r")
    txt.write("This is a test. Normal service will shortly resume!")
except IOError:
    print ("Error: unable to write the file. Check permissions")
else:
    print ("Content written to file successfully. Have a nice day.")
    txt.close()
```

STEP 7 Obviously this won't work due to the file textfile.txt being opened as read only (the "r" part). So in this case rather than Python telling you that you're doing something wrong, you've created an exception using the IOError class informing the user that the permissions are incorrect.



```
try:
    txt = open("/home/pi/Documents/textfile.txt", "r")
    txt.write("This is a test. Normal service will shortly resume!")
except IOError:
    print ("Error: unable to write the file. Check permissions")
else:
    print ("Content written to file successfully. Have a nice day.")
    txt.close()
```

STEP 8 Naturally, you can quickly fix the issue by changing the "r" read only instance with a "w" for write. This, as you already know, will create the file and write the content then commit the changes to the file. The end result will report a different set of circumstances, in this case, a successful execution of the code.



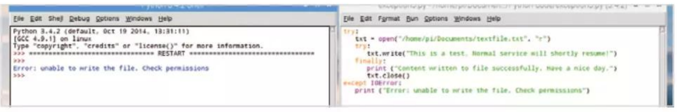
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>> 
Error: unable to write the file. Check permissions
>>> ================================ RESTART ================================
>>> 
Content written to file successfully. Have a nice day.
>>> 
```

```
try:
    txt = open("/home/pi/Documents/textfile.txt", "w")
    txt.write("This is a test. Normal service will shortly resume
except IOError:
    print ("Error: unable to write the file. Check permissions")
else:
    print ("Content written to file successfully. Have a nice day.
    txt.close()
```

```
This is a test. Normal service will shortly resume!
```

STEP 9 You can also use a finally: block, which works in a similar fashion but you can't use else with it. To use our example from Step 6:

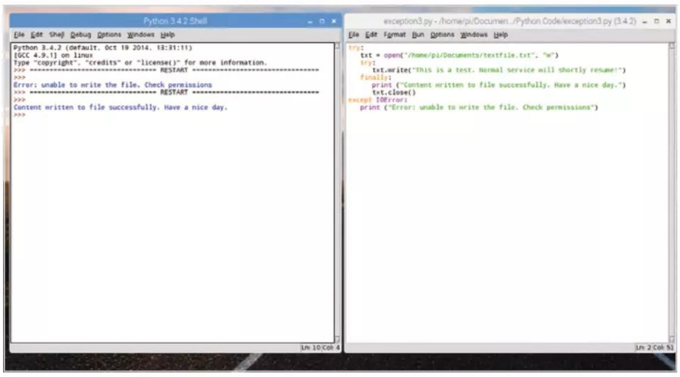
```
try:
 txt = open("/home/pi/Documents/textfile.txt", "r")
 try:
  txt.write("This is a test. Normal service will 
shortly resume!")
 finally:
  print ("Content written to file successfully. 
Have a nice day.")
  txt.close()
except IOError:
 print ("Error: unable to write the file. Check 
permissions")
```



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>> 
Error: unable to write the file. Check permissions
>>> 
```

```
try:
    txt = open("/home/pi/Documents/textfile.txt", "r")
    try:
        txt.write("This is a test. Normal service will shortly resume!")
    finally:
        print ("Content written to file successfully. Have a nice day.")
        txt.close()
except IOError:
    print ("Error: unable to write the file. Check permissions")
```

STEP 10 As before an error will occur as you've used the "r" read-only permission. If you change it to a "w", then the code will execute without the error being displayed in the IDLE Shell. Needless to say, it can be a tricky getting the exception code right the first time. Practise though, and you will get the hang of it.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>> 
Error: unable to write the file. Check permissions
>>> ================================ RESTART ================================
>>> 
Content written to file successfully. Have a nice day.
>>> 
```

```
try:
    txt = open("/home/pi/Documents/textfile.txt", "w")
    try:
        txt.write("This is a test. Normal service will shortly resume!")
    finally:
        print ("Content written to file successfully. Have a nice day.")
        txt.close()
except IOError:
    print ("Error: unable to write the file. Check permissions")
```

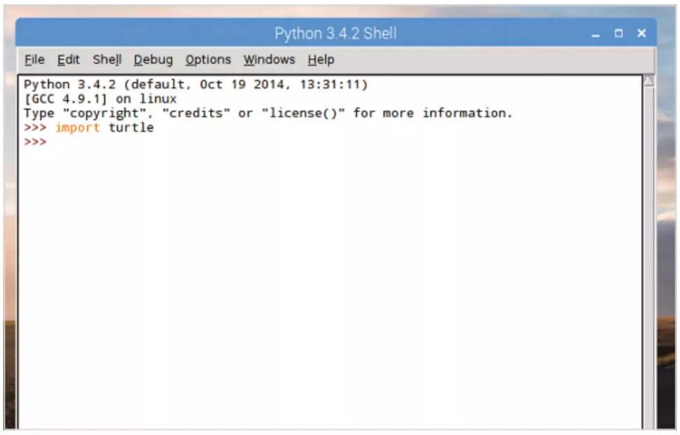
Python Graphics

While dealing with text on the screen, either as a game or in a program, is great, there will come a time when a bit of graphical representation wouldn't go amiss. Python 3 has numerous ways in which to include graphics and they're surprisingly powerful too.

GOING GRAPHICAL

You can draw simple graphics, lines, squares and so on, or you can use one of the many Python modules available, to bring out some spectacular effects.

STEP 1 One of the best graphical modules to begin learning Python graphics is Turtle. The Turtle module is, as the name suggests, based on the turtle robots used in many schools, that can be programmed to draw something on a large piece of paper on the floor. The Turtle module can be imported with: `import turtle`.

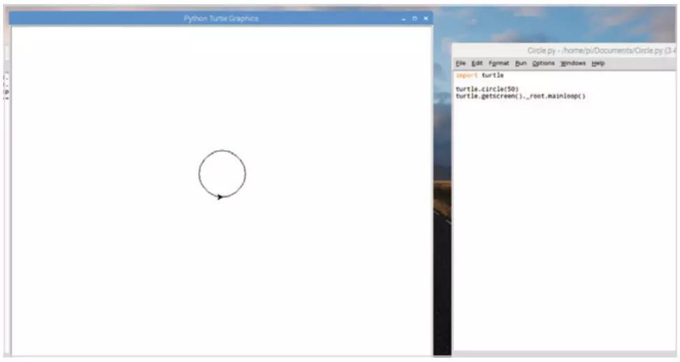

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import turtle
>>>
```

STEP 2 Let's begin by drawing a simple circle. Start a New File, then enter the following code:

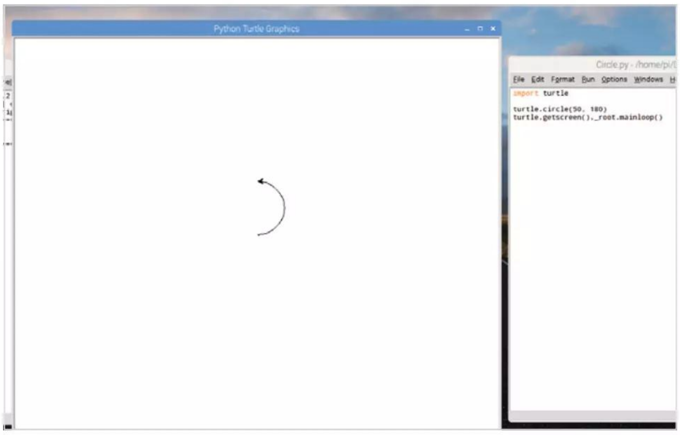
```
import turtle

turtle.circle(50)
turtle.getscreen()._root.mainloop()
```

As usual press F5 to save the code and execute it. A new window will now open up and the 'Turtle' will draw a circle.


Python Turtle Graphics
Circle.py - /home/pi/Documents/Circle.py (3.4
File Edit Format Run Options Windows Help

```
import turtle

turtle.circle(50)
turtle.getscreen()._root.mainloop()
```

STEP 3 The command turtle.circle(50) is what draws the circle on the screen, with 50 being the size. You can play around with the sizes if you like, going up to 100, 150 and beyond; you can draw an arc by entering: `turtle.circle(50, 180)`, where the size is 50, but you're telling Python to only draw 180° of the circle.

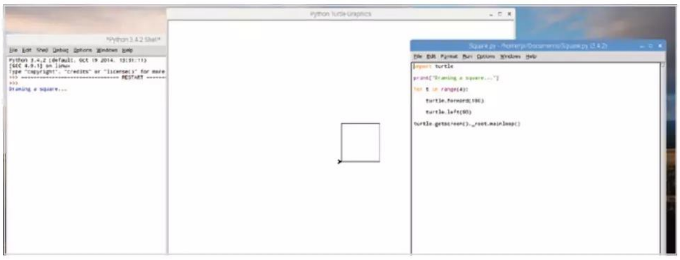

Python Turtle Graphics
Circle.py - /home/pi/
File Edit Format Run Options Windows H

```
import turtle

turtle.circle(50, 180)
turtle.getscreen()._root.mainloop()
```

STEP 4 The last part of the circle code tells Python to keep the window where the drawing is taking place to remain open, so the user can click to close it. Now, let's make a square:

```
import turtle

print(“Drawing a square…”)

for t in range(4):
   turtle.forward(100)
   turtle.left(90)
turtle.getscreen()._root.mainloop()
```

You can see that we've inserted a loop to draw the sides of the square.


Python Turtle Graphics

STEP 5

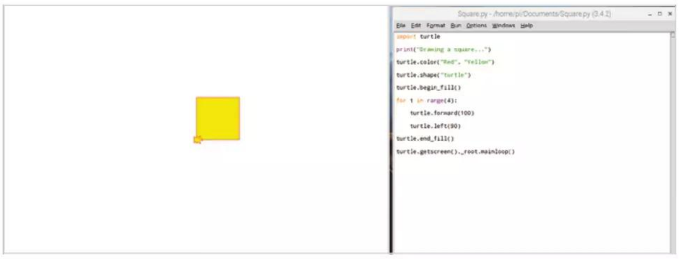
You can add a new line to the square code to add some colour:

```
turtle.color(“Red”)
```

Then you can even change the character to an actual turtle by entering:

```
turtle.shape(“turtle”)
```

You can also use the command `turtle.begin_fill()`, and `turtle.end_fill()` to fill in the square with the chosen colours; red outline, yellow fill in this case.


Square.py - /home/pi/Documents/Square.py (3.4.2)
File Edit Format Run Options Windows Help

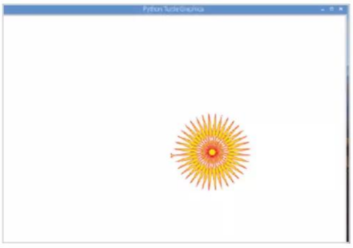
```
import turtle
print("Drawing a square...")
turtle.color("Red", "Yellow")
turtle.shape("turtle")
turtle.begin_fill()
for t in range(4):
    turtle.forward(100)
    turtle.left(90)
turtle.end_fill()
turtle.getscreen()._root.mainloop()
```

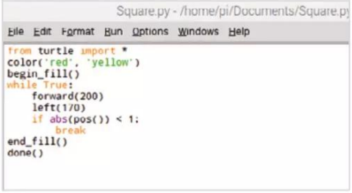
STEP 6

You can see that the Turtle module can draw out some pretty good shapes and become a little more complex as you begin to master the way it works. Enter this example:

```
from turtle import *
color(‘red’, ‘yellow’)
begin_fill()
while True:
  forward(200)
  left(170)
  if abs(pos()) < 1:
    break
end_fill()
done()
```

It’s a different method, but very effective.

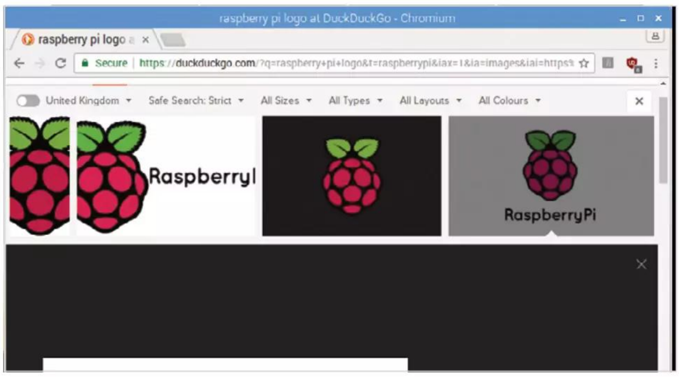

Python Turtle Graphics


Square.py - /home/pi/Documents/Square.py
File Edit Format Run Options Windows Help

```
from turtle import *
color('red', 'yellow')
begin_fill()
while True:
    forward(200)
    left(170)
    if abs(pos()) < 1:
        break
end_fill()
done()
```

STEP 7

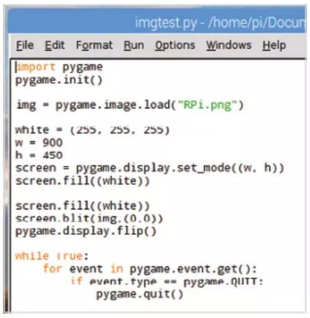
Another way in which you can display graphics is by using the Pygame module. There are numerous ways in which pygame can help you output graphics to the screen but for now let’s look at displaying a predefined image. Start by opening a browser and finding an image, then save it to the folder where you save your Python code.


raspberry pi logo at DuckDuckGo - Chromium
raspberry pi logo a
Secure
https://duckduckgo.com/?q=raspberry+pi+logo&t=raspberrypi&iax=1&ia=images&iai=https
United Kingdom
Safe Search: Strict
All Sizes
All Types
All Layouts
All Colours
Raspberry
RaspberryPi

STEP 8

Now let’s get the code by importing the pygame module:

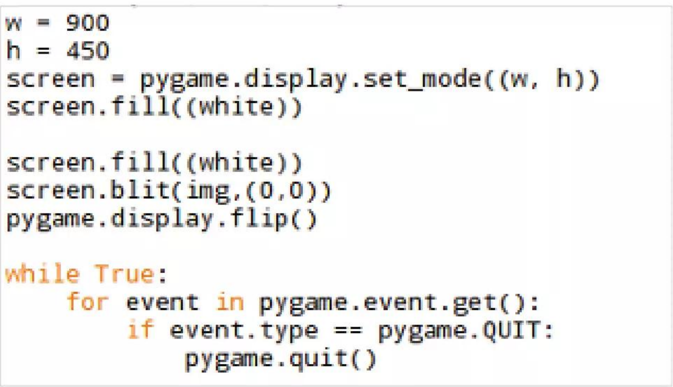
```
import pygame
pygame.init()

img = pygame.image.load(“RPi.png”)

white = (255, 255, 255)
w = 900
h = 450
screen = pygame.display.
set_mode((w, h))
screen.fill((white))

screen.fill((white))
screen.blit(img,(0,0))
pygame.display.flip()

while True:
 for event in pygame.event.get():
  if event.type == pygame.QUIT:
   pygame.quit()
```


imgtest.py - /home/pi/Docum
File Edit Format Run Options Windows Help

```
import pygame
pygame.init()

img = pygame.image.load("RPi.png")

white = (255, 255, 255)
w = 900
h = 450
screen = pygame.display.set_mode((w, h))
screen.fill((white))

screen.fill((white))
screen.blit(img,(0,0))
pygame.display.flip()

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
```

STEP 9

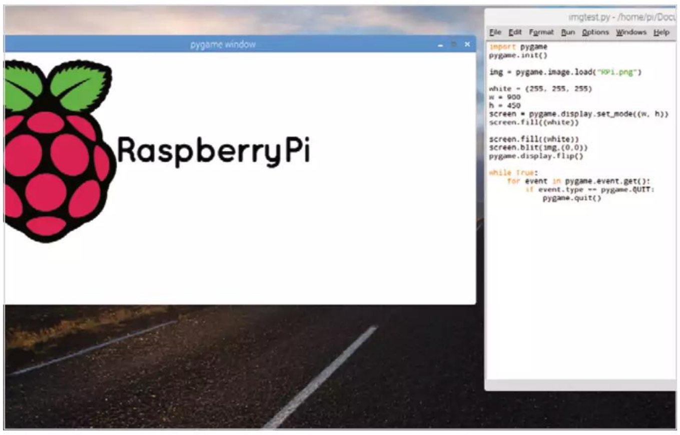
In the previous step you imported pygame, initiated the pygame engine and asked it to import our saved Raspberry Pi logo image, saved as RPi.png. Next you defined the background colour of the window to display the image and the window size as per the actual image dimensions. Finally you have a loop to close the window.



```
w = 900
h = 450
screen = pygame.display.set_mode((w, h))
screen.fill((white))

screen.fill((white))
screen.blit(img,(0,0))
pygame.display.flip()

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
```

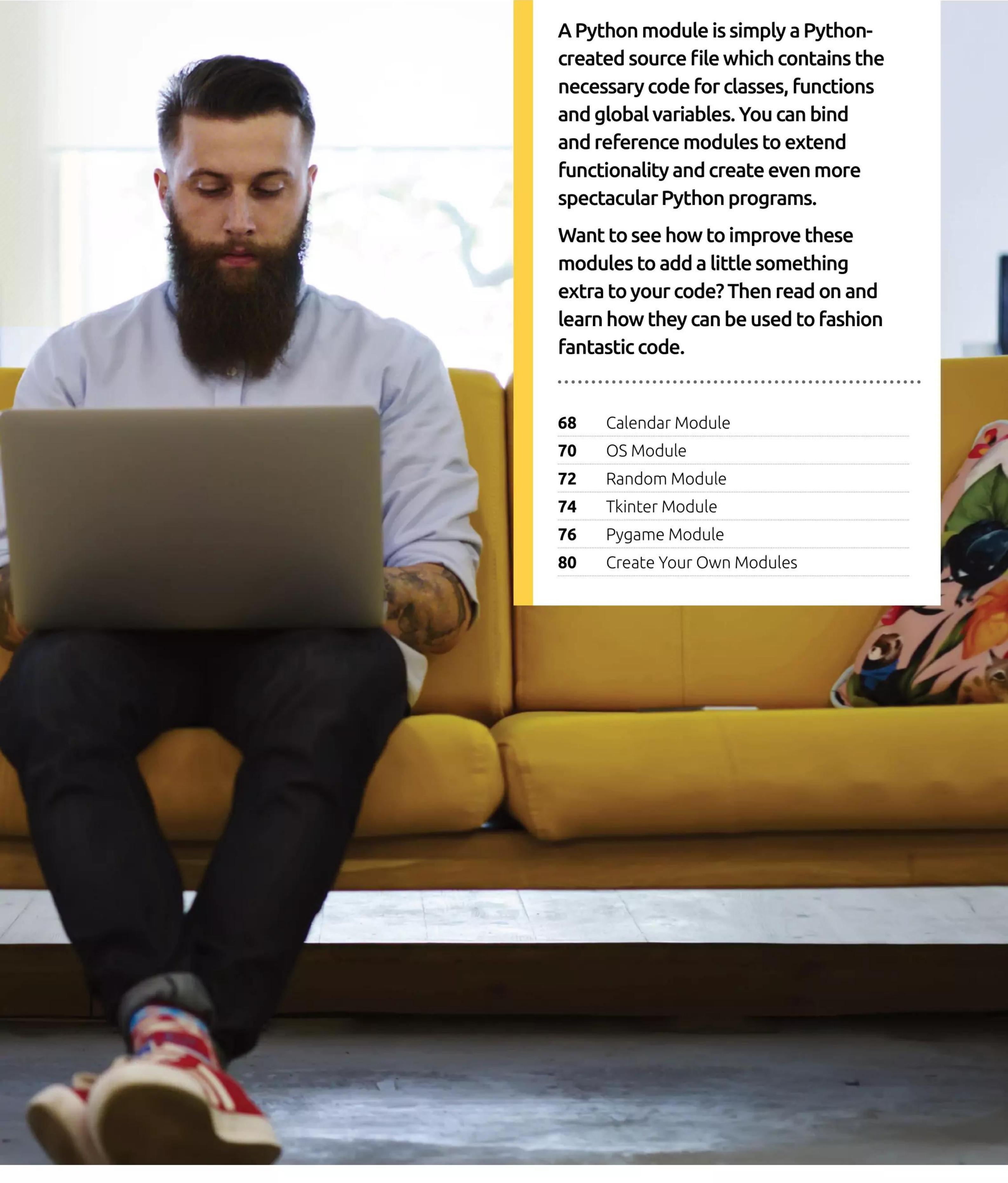
STEP 10

Press F5 to save and execute the code and your image will be displayed in a new window. Have a play around with the colours, sizes and so on and take time to look up the many functions within the pygame module too.


pygame window
RaspberryPi
imgtest.py - /home/pi/Docu
File Edit Format Run Options Windows Help

```
import pygame
pygame.init()

img = pygame.image.load("RPi.png")

white = (255, 255, 255)
w = 900
h = 450
screen = pygame.display.set_mode((w, h))
screen.fill((white))

screen.fill((white))
screen.blit(img,(0,0))
pygame.display.flip()

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
```

Using Modules



A Python module is simply a Python-created source file which contains the necessary code for classes, functions and global variables. You can bind and reference modules to extend functionality and create even more spectacular Python programs.

Want to see how to improve these modules to add a little something extra to your code? Then read on and learn how they can be used to fashion fantastic code.

68 Calendar Module
70 OS Module
72 Random Module
74 Tkinter Module
76 Pygame Module
80 Create Your Own Modules

Calendar Module

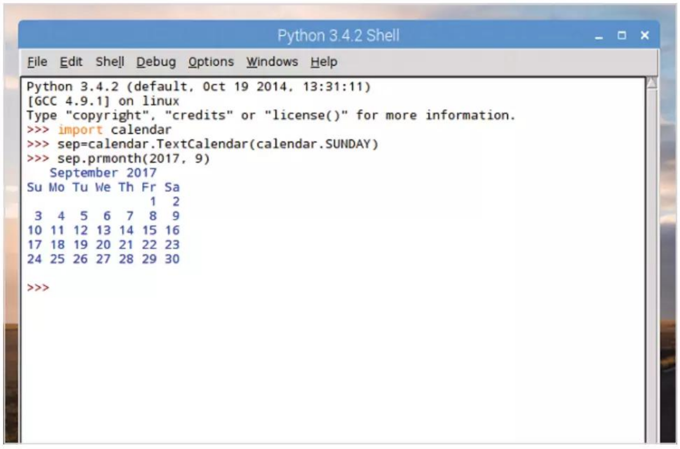
Beyond the time module, the calendar module can produce some interesting results when executed within your code. It does far more than simply display the date in the time module-like format, you can actually call up a wall calendar type display.

WORKING WITH DATES

The calendar module is built into Python 3. However, if for some reason it's not installed you can add it using pip install calendar as a Windows administrator or sudo pip install calendar for Linux and macOS.

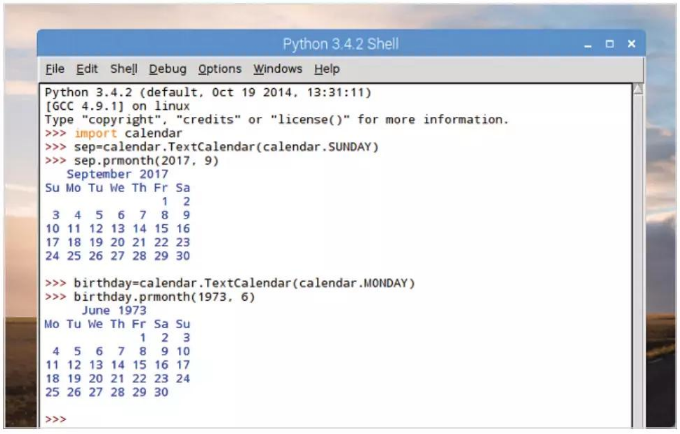
STEP 1 Launch Python 3 and enter: `import calendar` to call up the module and its inherent functions. Once it's loaded into memory, start by entering:

```
sep=calendar.TextCalendar(calendar.SUNDAY)
sep.prmonth(2017, 9)
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import calendar
>>> sep=calendar.TextCalendar(calendar.SUNDAY)
>>> sep.prmonth(2017, 9)
   September 2017
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30

>>>
```

STEP 2 You can see that the days of September 2017 are displayed in a wall calendar fashion. Naturally you can change the 2017, 9 part of the second line to any year and month you want, a birthday for example (1973, 6). The first line configures TextCalendar to start its weeks on a Sunday; you can opt for Monday if you prefer.

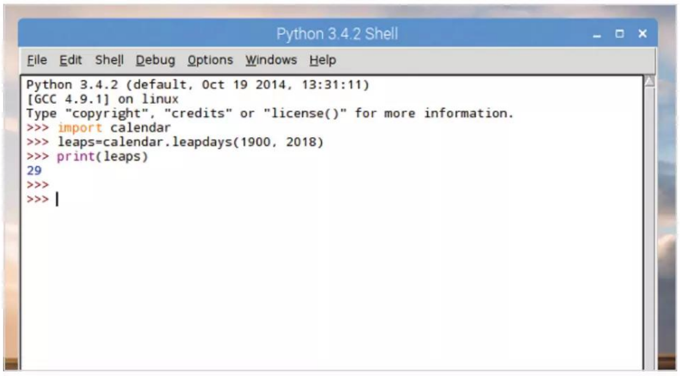

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import calendar
>>> sep=calendar.TextCalendar(calendar.SUNDAY)
>>> sep.prmonth(2017, 9)
   September 2017
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30

>>> birthday=calendar.TextCalendar(calendar.MONDAY)
>>> birthday.prmonth(1973, 6)
     June 1973
Mo Tu We Th Fr Sa Su
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30

>>>
```

STEP 3 There are numerous functions within the calendar module that may be of interest to you when forming your own code. For example, you can display the number of leap years between two specific years:

```
leaps=calendar.leapdays(1900, 2018)
print(leaps)
```

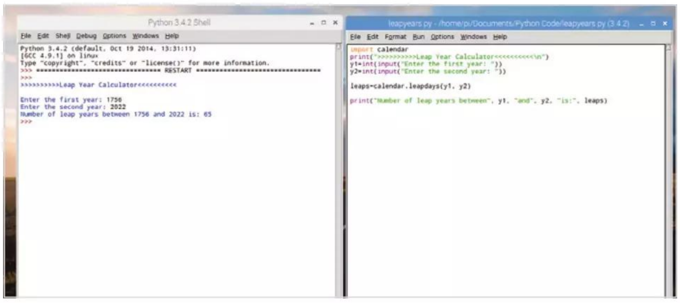
The result is 29, starting from 1904 onward.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import calendar
>>> leaps=calendar.leapdays(1900, 2018)
>>> print(leaps)
29
>>>
>>> |
```

STEP 4 You could even fashion that particular example into a piece of working, user interactive Python code:

```
import calendar
print(“>>>>>>>>>>Leap Year Calculator<<<<<<<<<<\n”)y1=int(input(“Enter the first year: ”))
y2=int(input(“Enter the second year: ”))
leaps=calendar.leapdays(y1, y2)
print(“Number of leap years between”, y1, “and”, y2, “is:”, leaps)
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
leapyears.py - /home/pi/Documents/Python Code/leapyears.py (3.4.2)
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
>>>>>>>>>>Leap Year Calculator<<<<<<<<<<

Enter the first year: 1756
Enter the second year: 2022
Number of leap years between 1756 and 2022 is: 65
>>>
```

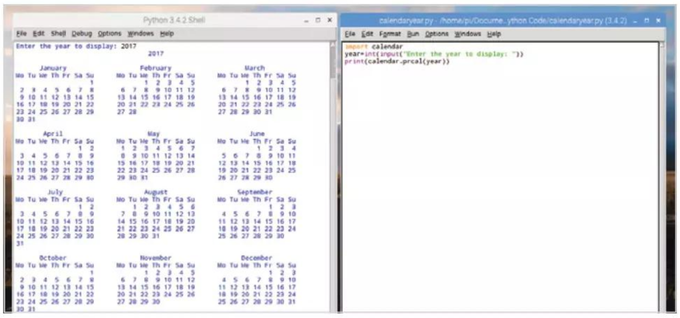
```
import calendar
print(">>>>>>>>>>Leap Year Calculator<<<<<<<<<<\n")
y1=int(input("Enter the first year: "))
y2=int(input("Enter the second year: "))

leaps=calendar.leapdays(y1, y2)

print("Number of leap years between", y1, "and", y2, "is:", leaps)
```

STEP 5

You can also create a program that will display all the days, weeks and months within a given year:

```
import calendar
year=int(input(“Enter the year to display: “)
print(calendar.prcal(year))
```

We’re sure you’ll agree that’s quite a handy bit of code to have to hand.



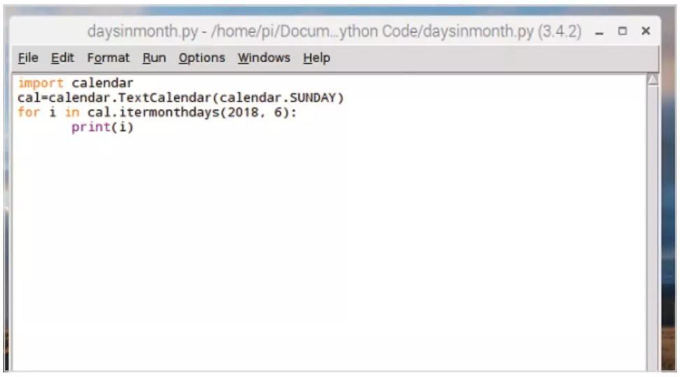
```
Enter the year to display: 2017
                                  2017

      January                   February                   March
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
                   1             1  2  3  4  5             1  2  3  4  5
 2  3  4  5  6  7  8       6  7  8  9 10 11 12       6  7  8  9 10 11 12
 9 10 11 12 13 14 15      13 14 15 16 17 18 19      13 14 15 16 17 18 19
16 17 18 19 20 21 22      20 21 22 23 24 25 26      20 21 22 23 24 25 26
23 24 25 26 27 28 29      27 28                     27 28 29 30 31
30 31

       April                      May                       June
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
                1  2       1  2  3  4  5  6  7                1  2  3  4
 3  4  5  6  7  8  9       8  9 10 11 12 13 14       5  6  7  8  9 10 11
10 11 12 13 14 15 16      15 16 17 18 19 20 21      12 13 14 15 16 17 18
17 18 19 20 21 22 23      22 23 24 25 26 27 28      19 20 21 22 23 24 25
24 25 26 27 28 29 30      29 30 31                  26 27 28 29 30

        July                     August                  September
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
                1  2          1  2  3  4  5  6                   1  2  3
 3  4  5  6  7  8  9       7  8  9 10 11 12 13       4  5  6  7  8  9 10
10 11 12 13 14 15 16      14 15 16 17 18 19 20      11 12 13 14 15 16 17
17 18 19 20 21 22 23      21 22 23 24 25 26 27      18 19 20 21 22 23 24
24 25 26 27 28 29 30      28 29 30 31               25 26 27 28 29 30
31

      October                   November                  December
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
                   1             1  2  3  4  5                   1  2  3
 2  3  4  5  6  7  8       6  7  8  9 10 11 12       4  5  6  7  8  9 10
 9 10 11 12 13 14 15      13 14 15 16 17 18 19      11 12 13 14 15 16 17
16 17 18 19 20 21 22      20 21 22 23 24 25 26      18 19 20 21 22 23 24
23 24 25 26 27 28 29      27 28 29 30               25 26 27 28 29 30 31
30 31
```

```
import calendar
year=int(input("Enter the year to display: "))
print(calendar.prcal(year))
```

STEP 6

Interestingly we can also list the number of days in a month by using a simple for loop:

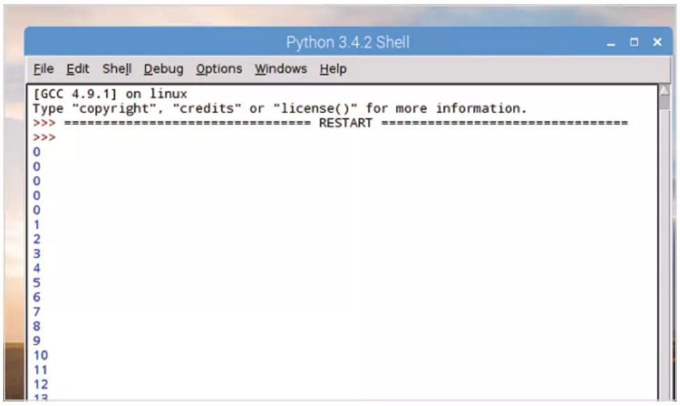
```
import calendar
cal=calendar.TextCalendar(calendar.SUNDAY)
for i in cal.itermonthdays(2018, 6):
   print(i)
```



```
import calendar
cal=calendar.TextCalendar(calendar.SUNDAY)
for i in cal.itermonthdays(2018, 6):
        print(i)
```

STEP 7

You can see that code produced some zeros at the beginning, this is due to the starting day of the week, Sunday in this case, and overlapping days from the previous month. So the counting of the days will start on Friday 1st June 2018 and will total 30 as the output correctly displays.



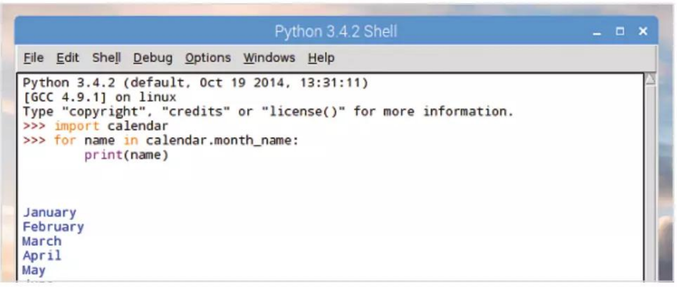
```
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
0
0
0
0
0
1
2
3
4
5
6
7
8
9
10
11
12
```

STEP 8

You’re also able to print the individual months or days of the week:

```
import calendar
for name in calendar.month_name:
   print(name)
```

```
import calendar
for name in calendar.day_name:
   print(name)
```



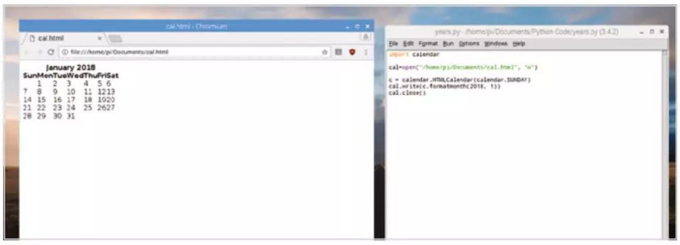
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import calendar
>>> for name in calendar.month_name:
        print(name)



January
February
March
April
May
```

STEP 9

The calendar module also allows us to write the functions in HTML, so that you can display it on a website. Let’s start by creating a new file:

```
import calendar
cal=open(“/home/pi/Documents/cal.html”, “w”)
c=calendar.HTMLCalendar(calendar.SUNDAY)
cal.write(c.formatmonth(2018, 1))
cal.close()
```

This code will create an HTML file called cal, open it with a browser and it displays the calendar for January 2018.



```
       January 2018
SunMonTueWedThuFriSat
    1   2   3    4   5 6
7   8   9   10   11  1213
14  15  16  17   18  1920
21  22  23  24   25  2627
28  29  30  31
```

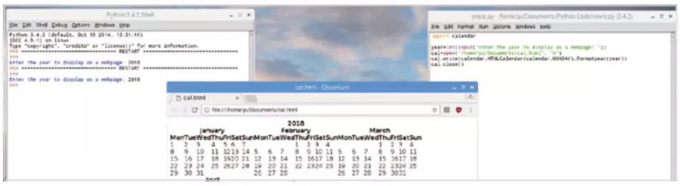
```
import calendar

cal=open("/home/pi/Documents/cal.html", "w")

c = calendar.HTMLCalendar(calendar.SUNDAY)
cal.write(c.formatmonth(2018, 1))
cal.close()
```

STEP 10

Of course, you can modify that to display a given year as a web page calendar:

```
import calendar
year=int(input(“Enter the year to display as a webpage: “))
cal=open(“/home/pi/Documents/cal.html”, “w”)
cal.write(calendar.HTMLCalendar(calendar.MONDAY).formatyear(year))
cal.close()
```

This code asks the user for a year, then creates the necessary webpage. Remember to change your file destination.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Enter the year to display as a webpage: 2018
>>> ================================ RESTART ================================
>>>
Enter the year to display as a webpage: 2018
>>>
```

OS Module

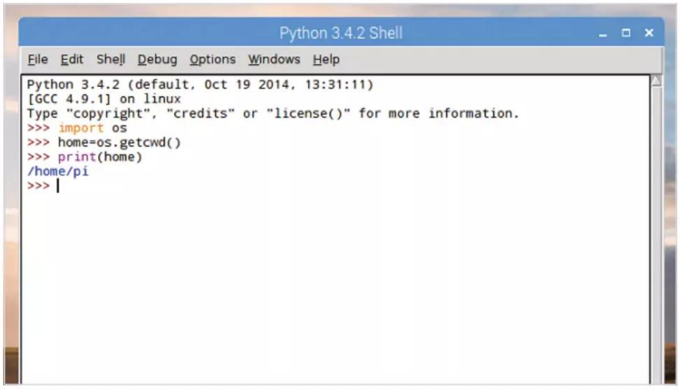
The OS module allows you to interact directly with the built-in commands found in your operating system. Commands vary depending on the OS you're running, as some will work with Windows whereas others will work with Linux and macOS.

INTO THE SYSTEM

One of the primary features of the OS module is the ability to list, move, create, delete and otherwise interact with files stored on the system, making it the perfect module for backup code.

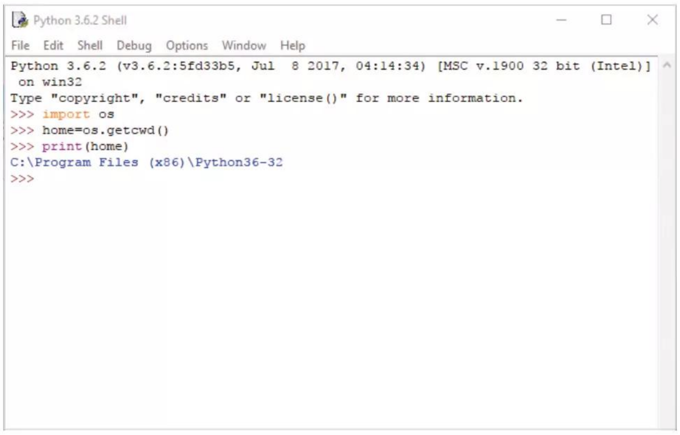
STEP 1 You can start the OS module with some simple functions to see how it interacts with the operating system environment that Python is running on. If you're using Linux or the Raspberry Pi, try this:

```
import os
home=os.getcwd()
print(home)
```



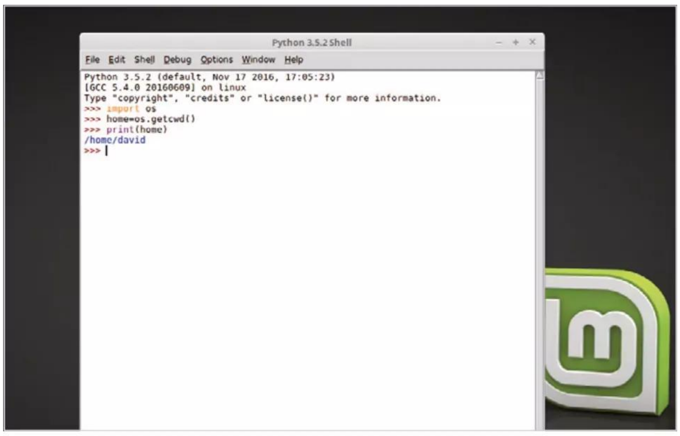
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import os
>>> home=os.getcwd()
>>> print(home)
/home/pi
>>> |
```

STEP 2 The returned result from printing the variable home is the current user's home folder on the system. In our example that's /home/pi; it will be different depending on the user name you log in as and the operating system you use. For example, Windows 10 will output: C:\Program Files (x86)\Python36-32.



```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> import os
>>> home=os.getcwd()
>>> print(home)
C:\Program Files (x86)\Python36-32
>>>
```

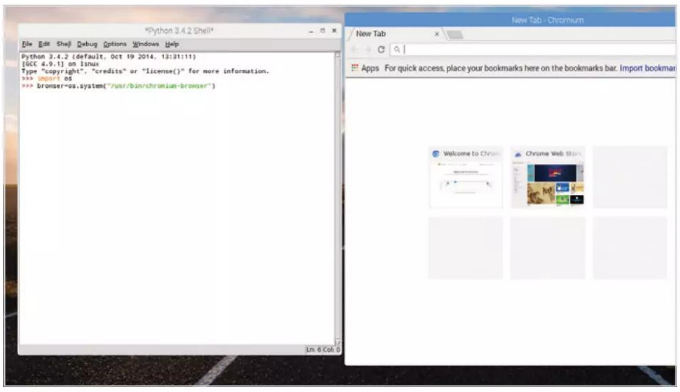
STEP 3 The Windows output is different as that's the current working directory of Python, as determined by the system; as you might suspect, the os.getcwd() function is asking Python to retrieve the Current Working Directory. Linux users will see something along the same lines as the Raspberry Pi, as will macOS users.



```
Python 3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import os
>>> home=os.getcwd()
>>> print(home)
/home/david
>>> |
```

STEP 4 Yet another interesting element to the OS module, is its ability to launch programs that are installed in the host system. For instance, if you wanted to launch the Chromium browser from within a Python program you can use the command:

```
import os
browser=os.system("/usr/bin/chromium-browser")
```

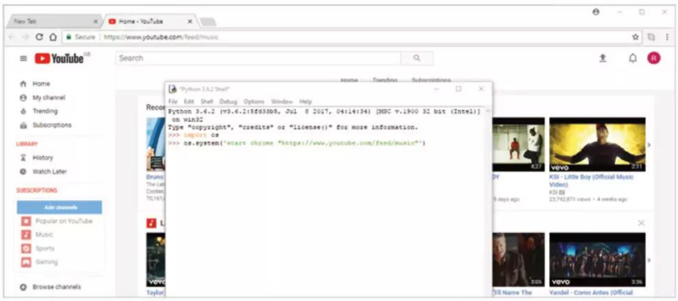


```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import os
>>> browser=os.system("/usr/bin/chromium-browser")
```

New Tab - Chromium
New Tab
Apps For quick access, place your bookmarks here on the bookmarks bar. Import bookmar
Welcome to Chro
Chrome Web Sto

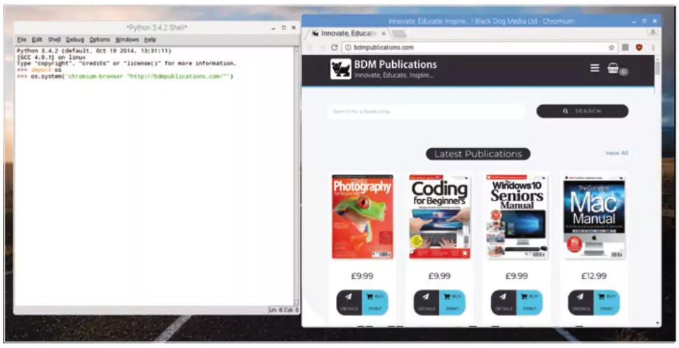
STEP 5 The os.system() function is what allows interaction with external programs; you can even call up previous Python programs using this method. You will obviously need to know the full path and program file name for it to work successfully. However, you can use the following:

```
import os
os.system(‘start chrome “https://www.youtube.com/
feed/music”’)
```



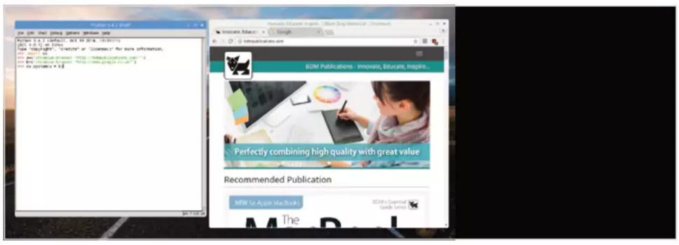
STEP 6 For Step 5’s example we used Windows, to show that the OS module works roughly the same across all platforms. In that case, we opened YouTube’s music feed page, so it is therefore possible to open specific pages:

```
import os
os.system(‘chromium-browser “http://
bdmpublications.com/”’)
```



STEP 7 Note in the previous step’s example the use of single and double-quotes. The single quotes encase the entire command and launching Chromium, whereas the double quotes open the specified page. You can even use variables to call multiple tabs in the same browser:

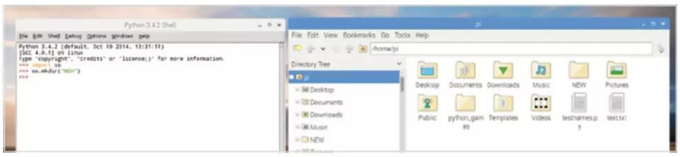
```
import os
a=(‘chromium-browser “http://bdmpublications.
com/”’)
b=(‘chromium-browser “http://www.google.co.uk”’)
os.system(a + b)
```



STEP 8 The ability to manipulate directories, or folders if you prefer, is one of the OS module’s best features. For example, to create a new directory you can use:

```
import os
os.mkdir(“NEW”)
```

This creates a new directory within the Current Working Directory, named according to the object in the mkdir function.

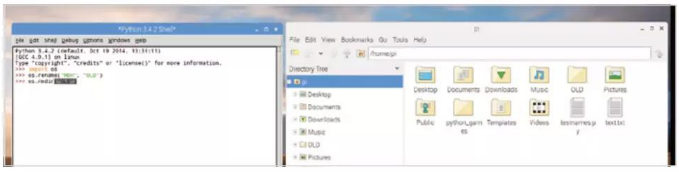


STEP 9 You can also rename any directories you’ve created by entering:

```
import os
os.rename(“NEW”, “OLD”)
```

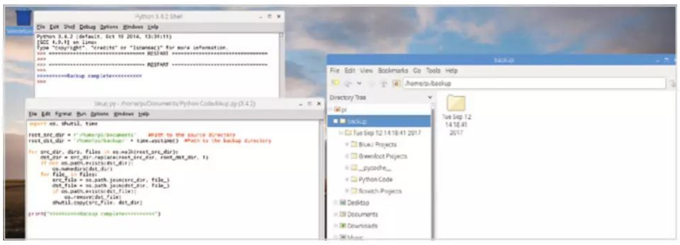
To delete them:

```
import os
os.rmdir(“OLD”)
```



STEP 10 Another module that goes together with OS is shutil. You can use the shutil module together with OS and time to create a time-stamped backup directory, and copy files into it:

```
import os, shutil, time

root_src_dir = r’/home/pi/Documents’
root_dst_dir = ‘/home/pi/backup/’ + time.asctime()

for src_dir, dirs, files in os.walk(root_src_dir):
  dst_dir = src_dir.replace(root_src_dir, root_
dst_dir, 1)
  if not os.path.exists(dst_dir):
    os.makedirs(dst_dir)
  for file_ in files:
    src_file = os.path.join(src_dir, file_)
    dst_file = os.path.join(dst_dir, file_)
    if os.path.exists(dst_file):
      os.remove(dst_file)
    shutil.copy(src_file, dst_dir)

print(“>>>>>>>>>>Backup complete<<<<<<<<<<”)
```



Random Module

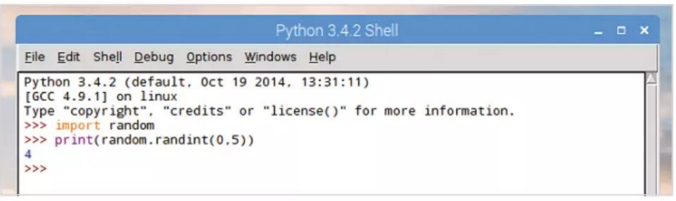
The random module is one you will likely come across many times in your Python programming lifetime; as the name suggests, it's designed to create random numbers or letters. However, it's not exactly random but it will suffice for most needs.

RANDOM NUMBERS

There are numerous functions within the random module, which when applied can create some interesting and very useful Python programs.

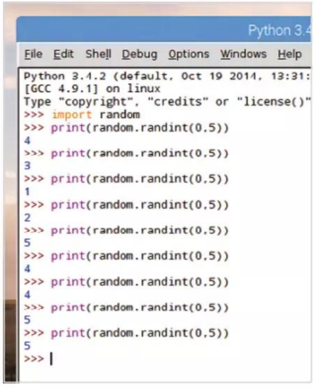
STEP 1 Just as with other modules you need to import random before you can use any of the functions we're going to look at in this tutorial. Let's begin by simply printing a random number from 1 to 5:

```
import random
print(randomint(0,5))
```



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import random
>>> print(random.randint(0,5))
4
>>>
```

STEP 2 In our example the number four was returned. However, enter the print function a few more times and it will display different integer values from the set of numbers given, zero to five. The overall effect, although pseudo-random, is adequate for the average programmer to utilise in their code.

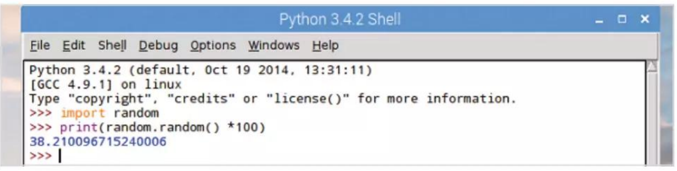


```
Python 3.4.2 (default, Oct 19 2014, 13:31:
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()"
>>> import random
>>> print(random.randint(0,5))
4
>>> print(random.randint(0,5))
3
>>> print(random.randint(0,5))
1
>>> print(random.randint(0,5))
2
>>> print(random.randint(0,5))
5
>>> print(random.randint(0,5))
4
>>> print(random.randint(0,5))
4
>>> print(random.randint(0,5))
5
>>> print(random.randint(0,5))
5
>>> |
```

STEP 3 For a bigger set of numbers, including floating point values, you can extend the range by using the multiplication sign:

```
import random
print(random.random() *100)
```

Will display a floating point number between 0 and 100, to the tune of around fifteen decimal points.

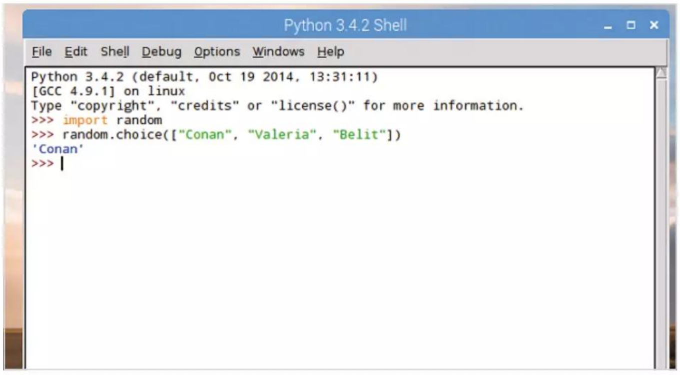


```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import random
>>> print(random.random() *100)
38.210096715240006
>>> |
```

STEP 4 However, the random module isn't used exclusively for numbers. You can use it to select an entry from a list from random, and the list can contain anything:

```
import random
random.choice([“Conan”, “Valeria”, “Belit”])
```

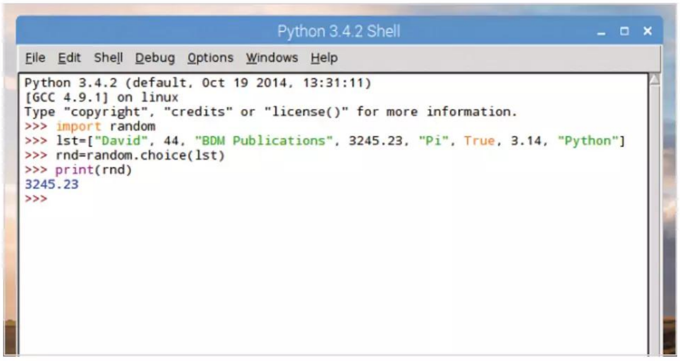
This will display one of the names of our adventurers at random, which is a great addition to a text adventure game.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import random
>>> random.choice(["Conan", "Valeria", "Belit"])
'Conan'
>>> |
```

STEP 5 You can extend the previous example somewhat by having random.choice() select from a list of mixed variables. For instance:

```
import random
lst=[“David”, 44, “BDM Publications”, 3245.23,
“Pi”, True, 3.14, “Python”]
rnd=random.choice(lst)
print(rnd)
```



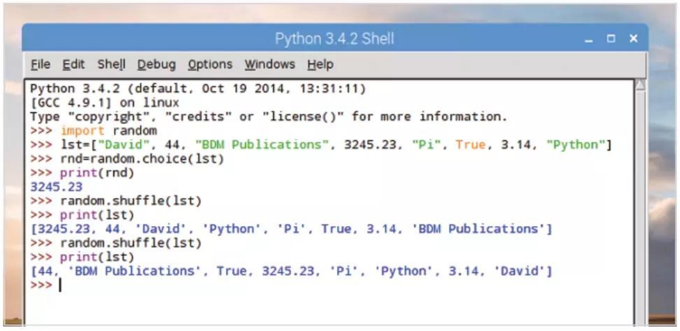
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import random
>>> lst=["David", 44, "BDM Publications", 3245.23, "Pi", True, 3.14, "Python"]
>>> rnd=random.choice(lst)
>>> print(rnd)
3245.23
>>>
```

STEP 6

Interestingly, you can also use a function within the random module to shuffle the items in the list, thus adding a little more randomness into the equation:

```
random.shuffle(lst)
print(lst)
```

This way, you can keep shuffling the list before displaying a random item from it.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

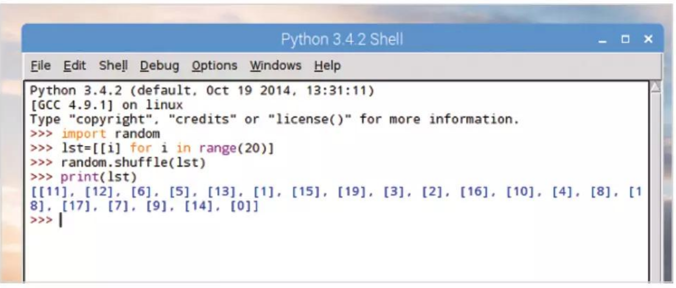
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import random
>>> lst=["David", 44, "BDM Publications", 3245.23, "Pi", True, 3.14, "Python"]
>>> rnd=random.choice(lst)
>>> print(rnd)
3245.23
>>> random.shuffle(lst)
>>> print(lst)
[3245.23, 44, 'David', 'Python', 'Pi', True, 3.14, 'BDM Publications']
>>> random.shuffle(lst)
>>> print(lst)
[44, 'BDM Publications', True, 3245.23, 'Pi', 'Python', 3.14, 'David']
>>> |
```

STEP 7

Using shuffle, you can create an entirely random list of numbers. For example, within a given range:

```
import random
lst=[[i] for I in range(20)]
random.shuffle(lst)
print(lst)
```

Keep shuffling the list and you can have a different selection of items from 0 to 20 every time.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

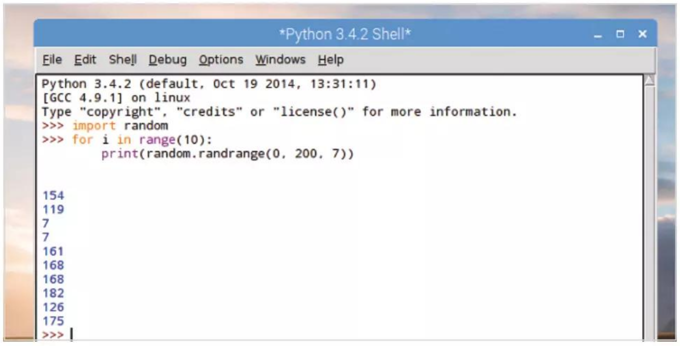
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import random
>>> lst=[[i] for i in range(20)]
>>> random.shuffle(lst)
>>> print(lst)
[[11], [12], [6], [5], [13], [1], [15], [19], [3], [2], [16], [10], [4], [8], [1
8], [17], [7], [9], [14], [0]]
>>> |
```

STEP 8

You can also select a random number from a given range in steps, using the start, stop, step loop:

```
import random
for i in range(10):
   print(random.randrange(0, 200, 7))
```

Results will vary but you get the general idea as to how it works.

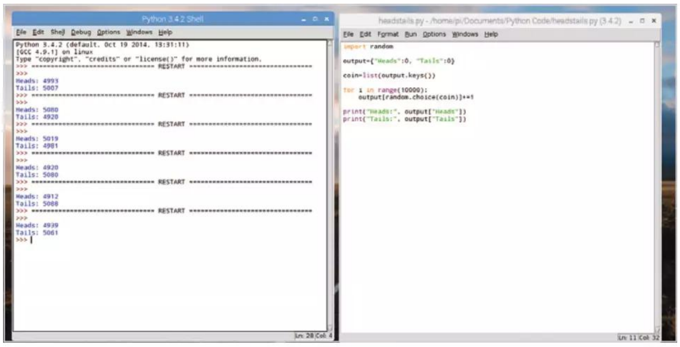

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import random
>>> for i in range(10):
        print(random.randrange(0, 200, 7))


154
119
7
7
161
168
168
182
126
175
>>> |
```

STEP 9

Let's use an example piece of code which flips a virtual coin ten thousand times and counts how many times it will land on heads or tails:

```
import random
output={"Heads":0, "Tails":0}
coin=list(output.keys())

for i in range(10000):
  output[random.choice(coin)]+=1

print("Heads:", output["Heads"])
print("Tails:", output["Tails"])
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Ln: 28 Col: 4
headstails.py - /home/pi/Documents/Python Code/headstails.py (3.4.2)
File Edit Format Run Options Windows Help
Ln: 11 Col: 32

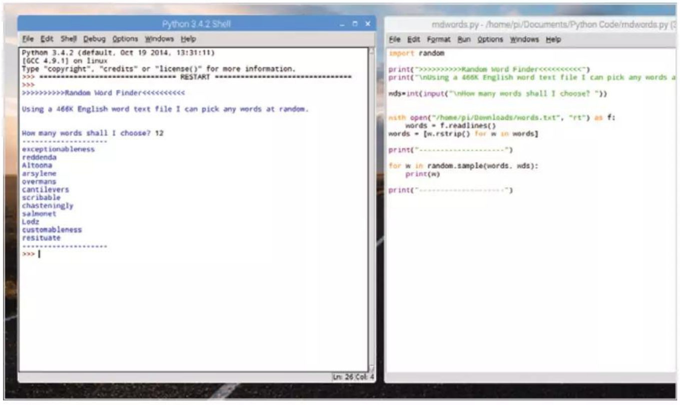
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Heads: 4993
Tails: 5007
>>> ================================ RESTART ================================
>>>
Heads: 5080
Tails: 4920
>>> ================================ RESTART ================================
>>>
Heads: 5019
Tails: 4981
>>> ================================ RESTART ================================
>>>
Heads: 4920
Tails: 5080
>>> ================================ RESTART ================================
>>>
Heads: 4912
Tails: 5088
>>> ================================ RESTART ================================
>>>
Heads: 4939
Tails: 5061
>>> |
```

```
import random

output={"Heads":0, "Tails":0}

coin=list(output.keys())

for i in range(10000):
    output[random.choice(coin)]+=1

print("Heads:", output["Heads"])
print("Tails:", output["Tails"])
```

STEP 10

Here's an interesting piece of code. Using a text file containing 466 thousand words, you can pluck a user generated number of words from the file (text file found at: **www.github.com/dwyl/english-words)**:

```
import random

print(">>>>>>>>>>Random Word Finder<<<<<<<<<<")
print("\nUsing a 466K English word text file I can
pick any words at random.\n")

wds=int(input("\nHow many words shall I choose?
"))

with open("/home/pi/Downloads/words.txt", "rt") as
f:
  words = f.readlines()
words = [w.rstrip() for w in words]

print("--------------------")

for w in random.sample(words, wds):
  print(w)

print("--------------------")
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Ln: 26 Col: 4
rndwords.py - /home/pi/Documents/Python Code/rndwords.py (3
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
>>>>>>>>>>Random Word Finder<<<<<<<<<<

Using a 466K English word text file I can pick any words at random.


How many words shall I choose? 12
--------------------
exceptionableness
reddenda
Altoona
arsylene
overmans
cantilevers
scribable
chasteningly
salmonet
Lodz
customableness
resituate
--------------------
>>> |
```

```
import random

print(">>>>>>>>>>Random Word Finder<<<<<<<<<<")
print("\nUsing a 466K English word text file I can pick any words a

wds=int(input("\nHow many words shall I choose? "))


with open("/home/pi/Downloads/words.txt", "rt") as f:
    words = f.readlines()
words = [w.rstrip() for w in words]

print("--------------------")

for w in random.sample(words, wds):
    print(w)

print("--------------------")
```

Tkinter Module

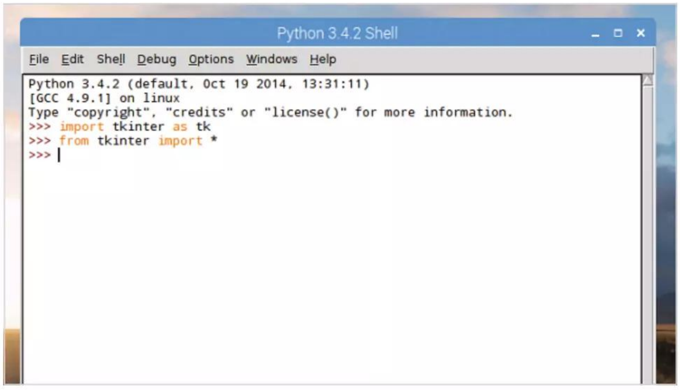
While running your code from the command line, or even in the Shell, is perfectly fine, Python is capable of so much more. The Tkinter module enables the programmer to set up a Graphical User Interface to interact with the user, and it's surprisingly powerful too.

GETTING GUI

Tkinter is easy to use but there's a lot more you can do with it. Let's start by seeing how it works and getting some code into it. Before long you will discover just how powerful this module really is.

STEP 1 Tkinter is usually built into Python 3. However, if it's available when you enter: `import tkinter`, then you need to `pip install tkinter` from the command prompt. We can start to import modules differently than before, to save on typing and by importing all their contents:

```
import tkinter as tk
from tkinter import *
```

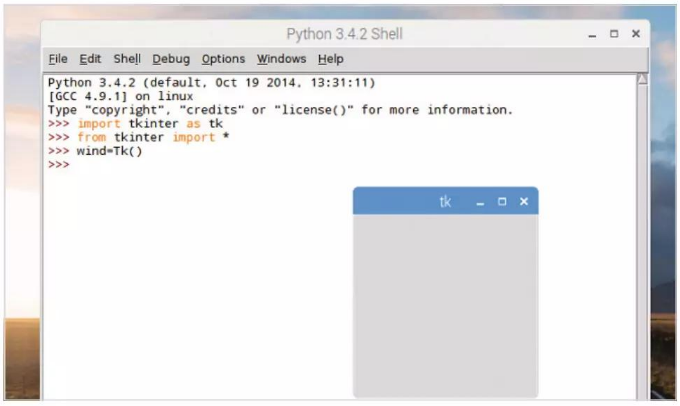

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import tkinter as tk
>>> from tkinter import *
>>> |
```

STEP 2 It's not recommended to import everything from a module using the asterisk but it won't do any harm normally. Let's begin by creating a basic GUI window, enter:

```
wind=Tk()
```

This creates a small, basic window. There's not much else to do at this point but click the X in the corner to close the window.

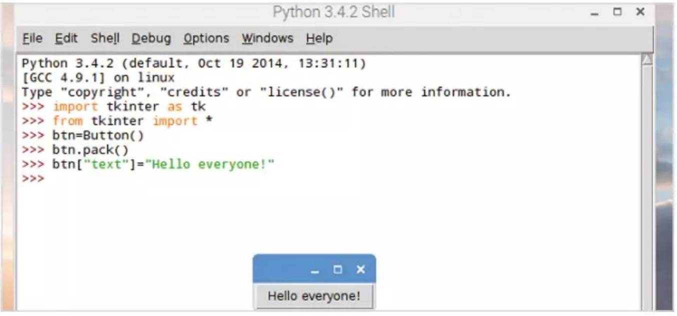

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
tk

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import tkinter as tk
>>> from tkinter import *
>>> wind=Tk()
>>>
```

STEP 3 The ideal approach is to add mainloop() into the code to control the Tkinter event loop, but we'll get to that soon. You've just created a Tkinter widget and there are several more we can play around with:

```
btn=Button()
btn.pack()
btn["text"]="Hello everyone!"
```

The first line focuses on the newly created window. Click back into the Shell and continue the other lines.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Hello everyone!

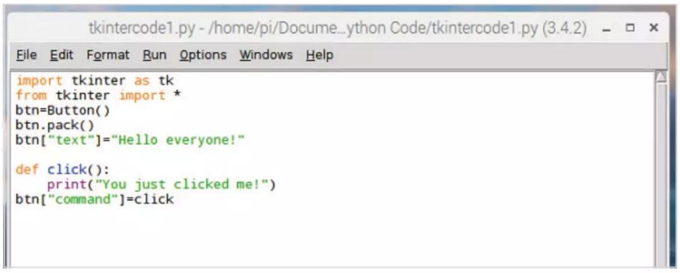
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import tkinter as tk
>>> from tkinter import *
>>> btn=Button()
>>> btn.pack()
>>> btn["text"]="Hello everyone!"
>>>
```

STEP 4 You can combine the above into a New File:

```
import tkinter as tk
from tkinter import *
btn=Button()
btn.pack()
btn["text"]="Hello everyone!"
```

Then add some button interactions:

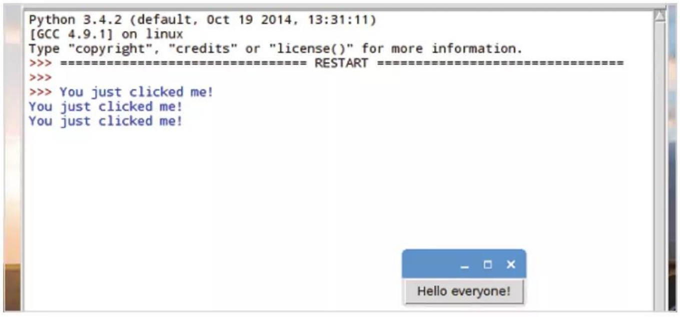
```
def click():
  print("You just clicked me!")
btn["command"]=click
```


tkintercode1.py - /home/pi/Docume...ython Code/tkintercode1.py (3.4.2)
File Edit Format Run Options Windows Help

```
import tkinter as tk
from tkinter import *
btn=Button()
btn.pack()
btn["text"]="Hello everyone!"

def click():
    print("You just clicked me!")
btn["command"]=click
```

STEP 5

Save and execute the code from Step 4 and a window appears with 'Hello everyone!' inside. If you click the Hello everyone! button, the Shell will output the text 'You just clicked me!'. It's simple but shows you what can be achieved with a few lines of code.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
>>> You just clicked me!
You just clicked me!
You just clicked me!
```

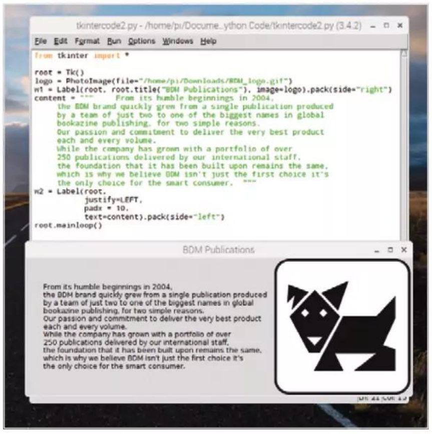
STEP 6

You can also display both text and images within a Tkinter window. However, only GIF, PGM or PPM formats are supported. So find an image and convert it before using the code. Here's an example using the BDM Publishing logo:

```
from tkinter import *

root = Tk()
logo = PhotoImage(file=”/home/pi/Downloads/BDM_logo.
gif”)
w1 = Label(root, root.title(“BDM Publications”),
image=logo).pack(side=”right”)
content = “””  From its humble beginnings in 2004,
the BDM brand quickly grew from a single publication
produced by a team of just two to one of the biggest
names in global bookazine publishing, for two simple
reasons. Our passion and commitment to deliver the
very best product each and every volume. While
the company has grown with a portfolio of over 250
publications delivered by our international staff,
the foundation that it has been built upon remains
the same, which is why we believe BDM isn’t just
the first choice it’s the only choice for the smart
consumer.  “””
w2 = Label(root,
    justify=LEFT,
    padx = 10,
    text=content).pack(side=”left”)
root.mainloop()
```

STEP 7

The previous code is quite weighty, mostly due to the content variable holding a part of BDM's About page from the company website. You can obviously change the content, the root.title and the image to suit your needs.



tkintercode2.py - /home/pi/Docume...ython Code/tkintercode2.py (3.4.2)

```
from tkinter import *

root = Tk()
logo = PhotoImage(file="/home/pi/Downloads/BDM_logo.gif")
w1 = Label(root, root.title("BDM Publications"), image=logo).pack(side="right")
content = """     From its humble beginnings in 2004,
         the BDM brand quickly grew from a single publication produced
         by a team of just two to one of the biggest names in global
         bookazine publishing, for two simple reasons.
         Our passion and commitment to deliver the very best product
         each and every volume.
         While the company has grown with a portfolio of over
         250 publications delivered by our international staff,
         the foundation that it has been built upon remains the same,
         which is why we believe BDM isn't just the first choice it's
         the only choice for the smart consumer.  """
w2 = Label(root,
           justify=LEFT,
           padx = 10,
           text=content).pack(side="left")
root.mainloop()
```

BDM Publications

From its humble beginnings in 2004,
the BDM brand quickly grew from a single publication produced
by a team of just two to one of the biggest names in global
bookazine publishing, for two simple reasons.
Our passion and commitment to deliver the very best product
each and every volume.
While the company has grown with a portfolio of over
250 publications delivered by our international staff,
the foundation that it has been built upon remains the same,
which is why we believe BDM isn't just the first choice it's
the only choice for the smart consumer.

STEP 8

You can create radio buttons too. Try:

```
from tkinter import *

root = Tk()

v = IntVar()

Label(root, root.title(“Options”), text=”””Choose
a preferred language:”””,
  justify = LEFT, padx = 20).pack()
Radiobutton(root,
    text=”Python”,
    padx = 20,
    variable=v,
    value=1).pack(anchor=W)
Radiobutton(root,
    text=”C++”,
    padx = 20,
    variable=v,
    value=2).pack(anchor=W)

mainloop()
```

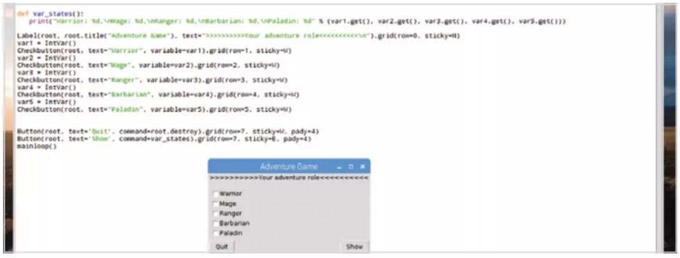
STEP 9

You can also create check boxes, with buttons and output to the Shell:

```
from tkinter import *
root = Tk()

def var_states():
  print(“Warrior: %d,\nMage: %d” % (var1.get(),
var2.get()))

Label(root, root.title(“Adventure Game”),
text=”>>>>>>>>>>Your adventure role<<<<<<<<<<”).
grid(row=0, sticky=N)
var1 = IntVar()
Checkbutton(root, text=”Warrior”, variable=var1).
grid(row=1, sticky=W)
var2 = IntVar()
Checkbutton(root, text=”Mage”, variable=var2).
grid(row=2, sticky=W)
Button(root, text=’Quit’, command=root.destroy).
grid(row=3, sticky=W, pady=4)
Button(root, text=’Show’, command=var_states).
grid(row=3, sticky=E, pady=4)

mainloop()
```

STEP 10

The code from Step 9 introduced some new geometry elements into Tkinter. Note the sticky=N, E and W arguments. These describe the locations of the check boxes and buttons (North, East, South and West). The row argument places them on separate rows. Have a play around and see what you get.



```
def var_states():
   print("Warrior: %d,\nMage: %d,\nRanger: %d,\nBarbarian: %d,\nPaladin: %d" % (var1.get(), var2.get(), var3.get(), var4.get(), var5.get()))

Label(root, root.title("Adventure Game"), text=">>>>>>>>>>Your adventure role<<<<<<<<<<\n").grid(row=0, sticky=N)
var1 = IntVar()
Checkbutton(root, text="Warrior", variable=var1).grid(row=1, sticky=W)
var2 = IntVar()
Checkbutton(root, text="Mage", variable=var2).grid(row=2, sticky=W)
var3 = IntVar()
Checkbutton(root, text="Ranger", variable=var3).grid(row=3, sticky=W)
var4 = IntVar()
Checkbutton(root, text="Barbarian", variable=var4).grid(row=4, sticky=W)
var5 = IntVar()
Checkbutton(root, text="Paladin", variable=var5).grid(row=5, sticky=W)


Button(root, text='Quit', command=root.destroy).grid(row=7, sticky=W, pady=4)
Button(root, text='Show', command=var_states).grid(row=7, sticky=E, pady=4)
mainloop()
```

Adventure Game

>>>>>>>>>>Your adventure role<<<<<<<<<<

☐ Warrior
☐ Mage
☐ Ranger
☐ Barbarian
☐ Paladin

Pygame Module

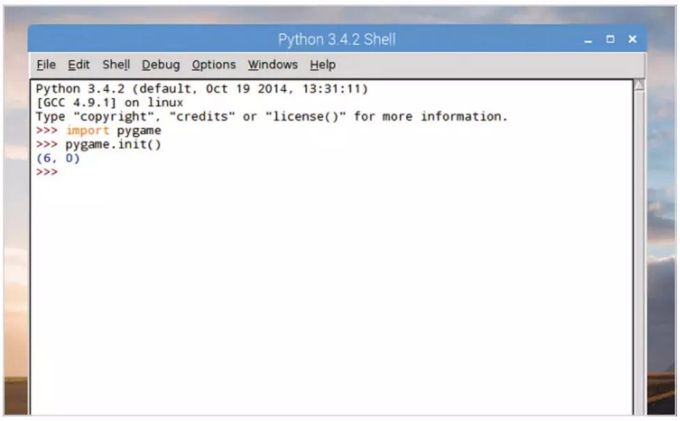
We've had a brief look at the Pygame module already but there's a lot more to it that needs exploring. Pygame was developed to help Python programmers create either graphical or text-based games.

PYGAMING

Pygame isn't an inherent module to Python but those using the Raspberry Pi will already have it installed. Everyone else will need to use: pip install pygame from the command prompt.

STEP 1 Naturally you need to load up the Pygame modules into memory before you're able to utilise them. Once that's done Pygame requires the user to initialise it prior to any of the functions being used:

```
import pygame
pygame.init()
```

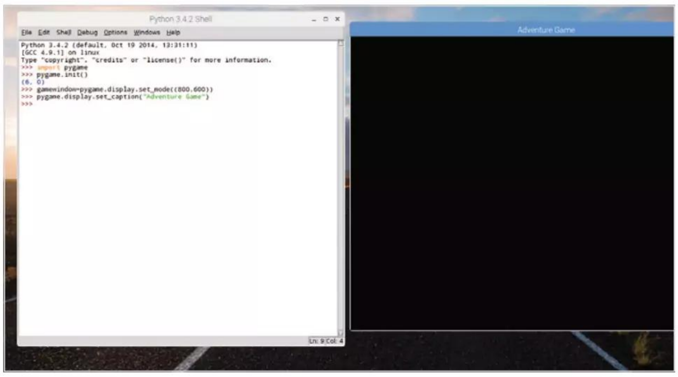


```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import pygame
>>> pygame.init()
(6, 0)
>>>
```

STEP 2 Let's create a simple game ready window, and give it a title:

```
gamewindow=pygame.display.set_mode((800,600))
pygame.display.set_caption("Adventure Game")
```

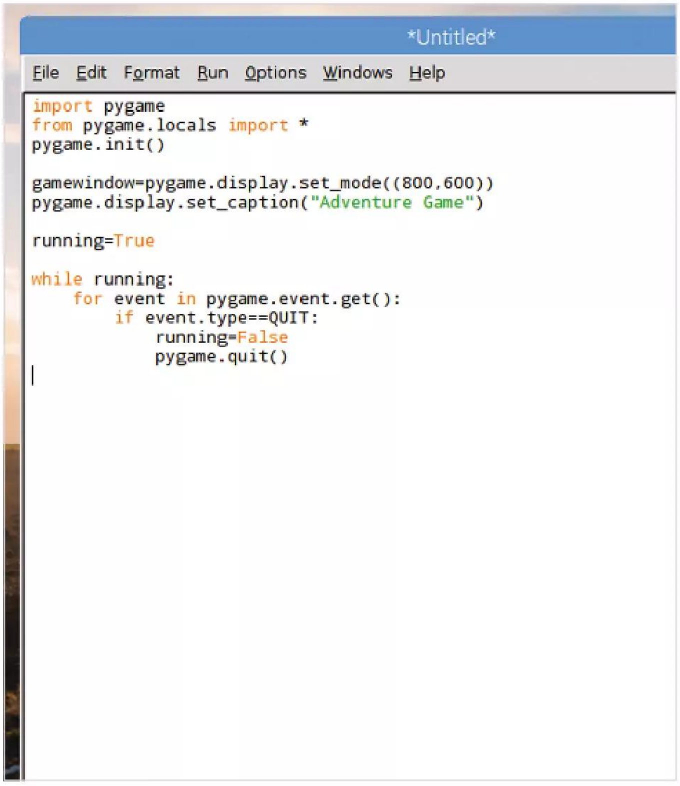
You can see that after the first line is entered, you need to click back into the IDLE Shell to continue entering code; also, you can change the title of the window to anything you like.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import pygame
>>> pygame.init()
(6, 0)
>>> gamewindow=pygame.display.set_mode((800,600))
>>> pygame.display.set_caption("Adventure Game")
>>>
```

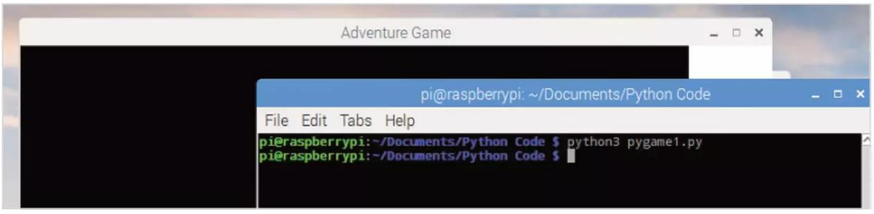
STEP 3 Sadly you can't close the newly created Pygame window without closing the Python IDLE Shell, which isn't very practical. For this reason, you need to work in the editor (New > File) and create a True/False while loop:

```
import pygame
from pygame.locals import *
pygame.init()

gamewindow=pygame.display.set_mode((800,600))
pygame.display.set_caption("Adventure Game")

running=True

while running:
  for event in pygame.event.get():
   if event.type==QUIT:
     running=False
     pygame.quit()
```



```
import pygame
from pygame.locals import *
pygame.init()

gamewindow=pygame.display.set_mode((800,600))
pygame.display.set_caption("Adventure Game")

running=True

while running:
    for event in pygame.event.get():
        if event.type==QUIT:
            running=False
            pygame.quit()
```

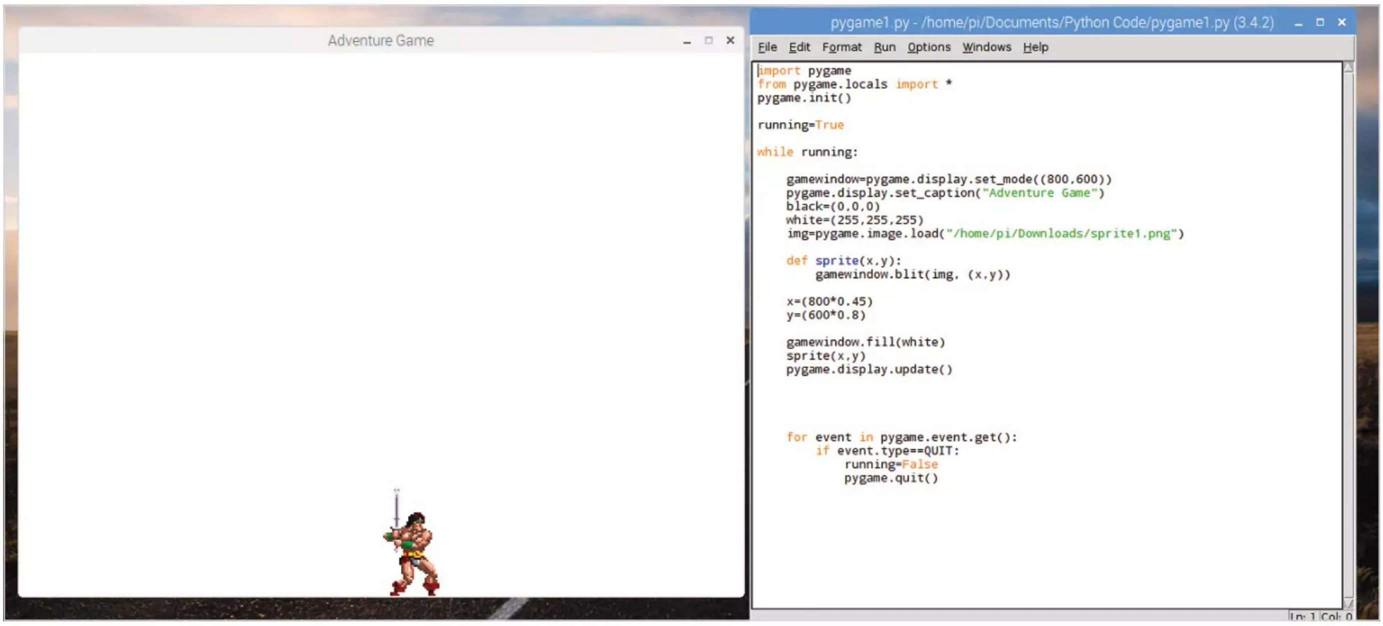
STEP 4 If the Pygame window still won't close don't worry, it's just a discrepancy between the IDLE (which is written with Tkinter) and the Pygame module. If you run your code via the command line, it closes perfectly well.



```
pi@raspberrypi:~/Documents/Python Code $ python3 pygame1.py
pi@raspberrypi:~/Documents/Python Code $ 
```

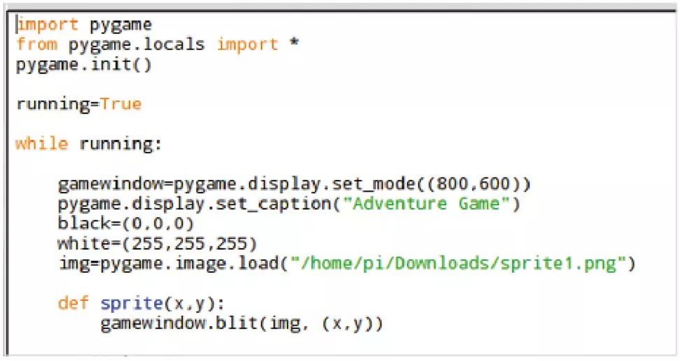
STEP 5 You're going to shift the code around a bit now, running the main Pygame code within a while loop; it makes it neater and easier to follow. We've downloaded a graphic to use and we need to set some parameters for pygame:

```
import pygame
pygame.init()

running=True

while running:

  gamewindow=pygame.display.set_mode((800,600))
  pygame.display.set_caption(“Adventure Game”)
  black=(0,0,0)
  white=(255,255,255)
  img=pygame.image.load(“/home/pi/Downloads/sprite1.png”)

  def sprite(x,y):
    gamewindow.blit(img, (x,y))

  x=(800*0.45)
  y=(600*0.8)

  gamewindow.fill(white)
  sprite(x,y)
  pygame.display.update()

  for event in pygame.event.get():
    if event.type==pygame.QUIT:
      running=False
```

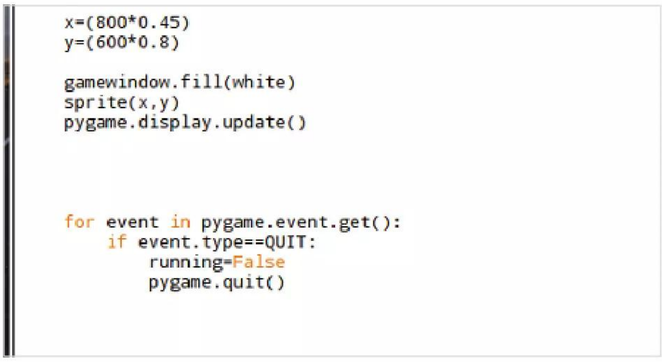

Adventure Game
pygame1.py - /home/pi/Documents/Python Code/pygame1.py (3.4.2)
File Edit Format Run Options Windows Help
Ln: 1 Col: 0

```
import pygame
from pygame.locals import *
pygame.init()

running=True

while running:

    gamewindow=pygame.display.set_mode((800,600))
    pygame.display.set_caption("Adventure Game")
    black=(0,0,0)
    white=(255,255,255)
    img=pygame.image.load("/home/pi/Downloads/sprite1.png")

    def sprite(x,y):
        gamewindow.blit(img, (x,y))

    x=(800*0.45)
    y=(600*0.8)

    gamewindow.fill(white)
    sprite(x,y)
    pygame.display.update()


    for event in pygame.event.get():
        if event.type==QUIT:
            running=False
            pygame.quit()
```

STEP 6 Let's quickly go through the code changes. We've defined two colours, black and white together with their respective RGB colour values. Next we've loaded the downloaded image called sprite1.png and allocated it to the variable img; and also defined a sprite function and the Blit function will allow us to eventually move the image.



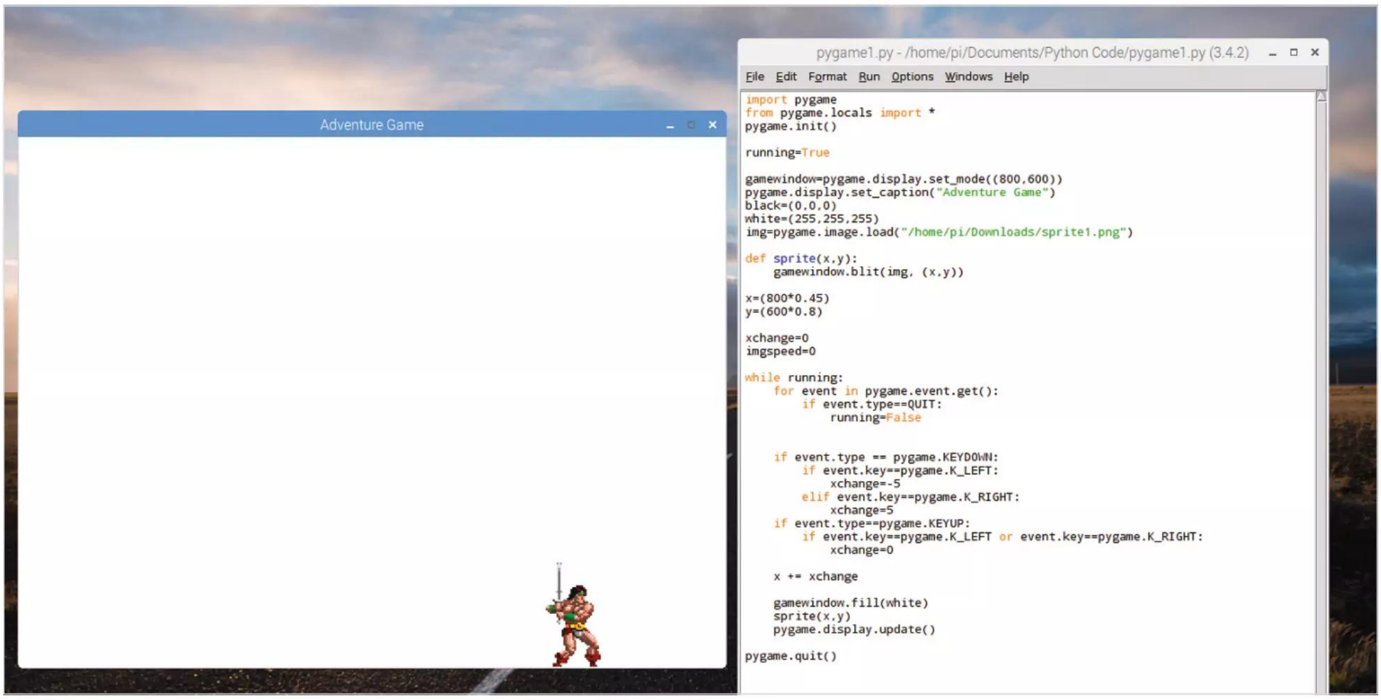
```
import pygame
from pygame.locals import *
pygame.init()

running=True

while running:

    gamewindow=pygame.display.set_mode((800,600))
    pygame.display.set_caption("Adventure Game")
    black=(0,0,0)
    white=(255,255,255)
    img=pygame.image.load("/home/pi/Downloads/sprite1.png")

    def sprite(x,y):
        gamewindow.blit(img, (x,y))
```



```
    x=(800*0.45)
    y=(600*0.8)

    gamewindow.fill(white)
    sprite(x,y)
    pygame.display.update()



    for event in pygame.event.get():
        if event.type==QUIT:
            running=False
            pygame.quit()
```

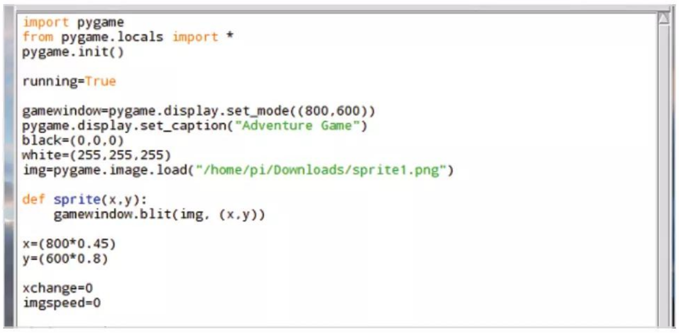
STEP 7 Now we can change the code around again, this time containing a movement option within the while loop, and adding the variables needed to move the sprite around the screen:

```
import pygame
from pygame.locals import *
pygame.init()

running=True

gamewindow=pygame.display.set_mode((800,600))
pygame.display.set_caption("Adventure Game")
black=(0,0,0)
white=(255,255,255)
img=pygame.image.load("/home/pi/Downloads/sprite1.
png")

def sprite(x,y):
 gamewindow.blit(img, (x,y))

x=(800*0.45)
y=(600*0.8)

xchange=0
imgspeed=0

while running:
 for event in pygame.event.get():
  if event.type==QUIT:
   running=False

 if event.type == pygame.KEYDOWN:
  if event.key==pygame.K_LEFT:
   xchange=-5
  elif event.key==pygame.K_RIGHT:
   xchange=5
 if event.type==pygame.KEYUP:
  if event.key==pygame.K_LEFT or event
key==pygame.K_RIGHT:
   xchange=0

 x += xchange

 gamewindow.fill(white)
 sprite(x,y)
 pygame.display.update()

pygame.quit()
```

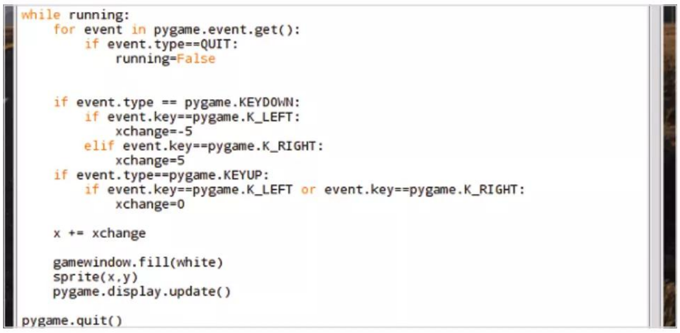

Adventure Game
pygame1.py - /home/pi/Documents/Python Code/pygame1.py (3.4.2)
File Edit Format Run Options Windows Help

```
import pygame
from pygame.locals import *
pygame.init()

running=True

gamewindow=pygame.display.set_mode((800,600))
pygame.display.set_caption("Adventure Game")
black=(0,0,0)
white=(255,255,255)
img=pygame.image.load("/home/pi/Downloads/sprite1.png")

def sprite(x,y):
    gamewindow.blit(img, (x,y))

x=(800*0.45)
y=(600*0.8)

xchange=0
imgspeed=0

while running:
    for event in pygame.event.get():
        if event.type==QUIT:
            running=False


    if event.type == pygame.KEYDOWN:
        if event.key==pygame.K_LEFT:
            xchange=-5
        elif event.key==pygame.K_RIGHT:
            xchange=5
    if event.type==pygame.KEYUP:
        if event.key==pygame.K_LEFT or event.key==pygame.K_RIGHT:
            xchange=0

    x += xchange

    gamewindow.fill(white)
    sprite(x,y)
    pygame.display.update()

pygame.quit()
```

STEP 8 Copy the code down and using the left and right arrow keys on the keyboard you can move your sprite across the bottom of the screen. Now, it looks like you have the makings of a classic arcade 2D scroller in the works.



```
import pygame
from pygame.locals import *
pygame.init()

running=True

gamewindow=pygame.display.set_mode((800,600))
pygame.display.set_caption("Adventure Game")
black=(0,0,0)
white=(255,255,255)
img=pygame.image.load("/home/pi/Downloads/sprite1.png")

def sprite(x,y):
    gamewindow.blit(img, (x,y))

x=(800*0.45)
y=(600*0.8)

xchange=0
imgspeed=0
```

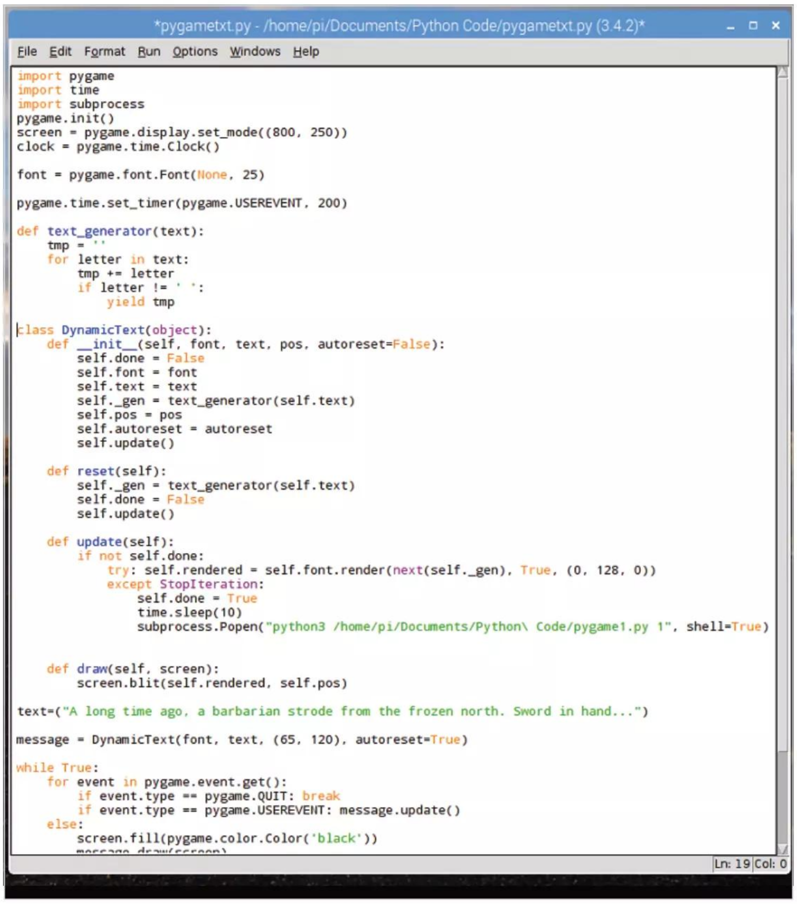


```
while running:
    for event in pygame.event.get():
        if event.type==QUIT:
            running=False


    if event.type == pygame.KEYDOWN:
        if event.key==pygame.K_LEFT:
            xchange=-5
        elif event.key==pygame.K_RIGHT:
            xchange=5
    if event.type==pygame.KEYUP:
        if event.key==pygame.K_LEFT or event.key==pygame.K_RIGHT:
            xchange=0

    x += xchange

    gamewindow.fill(white)
    sprite(x,y)
    pygame.display.update()

pygame.quit()
```

STEP 9

You can now implement a few additions and utilise some previous tutorial code. The new elements are the subprocess module, of which one function allows us to launch a second Python script from within another; and we're going to create a New File called pygametxt.py:

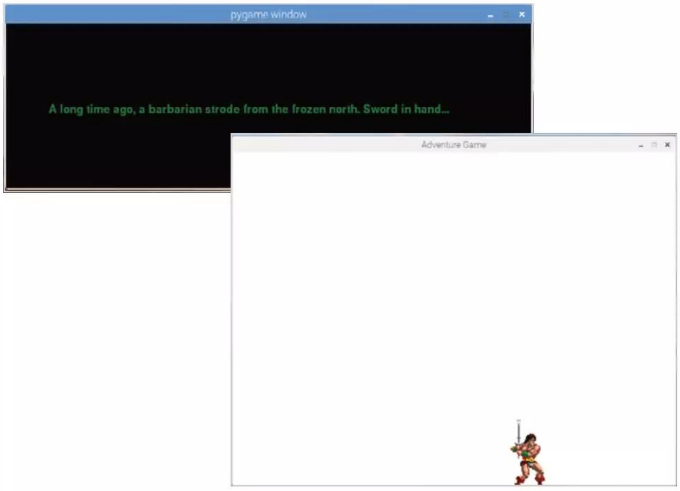
```
import pygame
import time
import subprocess
pygame.init()
screen = pygame.display.set_mode((800, 250))
clock = pygame.time.Clock()

font = pygame.font.Font(None, 25)

pygame.time.set_timer(pygame.USEREVENT, 200)

def text_generator(text):
  tmp = ‘’
  for letter in text:
    tmp += letter
    if letter != ‘ ‘:
      yield tmp

class DynamicText(object):
  def __init__(self, font, text, pos,
autoreset=False):
    self.done = False
    self.font = font
    self.text = text
    self._gen = text_generator(self.text)
    self.pos = pos
    self.autoreset = autoreset
    self.update()

  def reset(self):
    self._gen = text_generator(self.text)
    self.done = False
    self.update()

  def update(self):
    if not self.done:
      try: self.rendered = self.font.
render(next(self._gen), True, (0, 128, 0))
      except StopIteration:
        self.done = True
        time.sleep(10)
        subprocess.Popen(“python3 /home/pi/Documents/
Python\ Code/pygame1.py 1”, shell=True)

  def draw(self, screen):
    screen.blit(self.rendered, self.pos)

text=(“A long time ago, a barbarian strode from the
frozen north. Sword in hand...”)

message = DynamicText(font, text, (65, 120),
autoreset=True)

while True:
  for event in pygame.event.get():
    if event.type == pygame.QUIT: break
    if event.type == pygame.USEREVENT: message.
update()
  else:
    screen.fill(pygame.color.Color(‘black’))
    message.draw(screen)
    pygame.display.flip()
    clock.tick(60)
    continue
  break

pygame.quit()
```


pygametxt.py - /home/pi/Documents/Python Code/pygametxt.py (3.4.2)
File
Edit
Format
Run
Options
Windows
Help
Ln: 19
Col: 0

```
import pygame
import time
import subprocess
pygame.init()
screen = pygame.display.set_mode((800, 250))
clock = pygame.time.Clock()

font = pygame.font.Font(None, 25)

pygame.time.set_timer(pygame.USEREVENT, 200)

def text_generator(text):
    tmp = ''
    for letter in text:
        tmp += letter
        if letter != ' ':
            yield tmp

class DynamicText(object):
    def __init__(self, font, text, pos, autoreset=False):
        self.done = False
        self.font = font
        self.text = text
        self._gen = text_generator(self.text)
        self.pos = pos
        self.autoreset = autoreset
        self.update()

    def reset(self):
        self._gen = text_generator(self.text)
        self.done = False
        self.update()

    def update(self):
        if not self.done:
            try: self.rendered = self.font.render(next(self._gen), True, (0, 128, 0))
            except StopIteration:
                self.done = True
                time.sleep(10)
                subprocess.Popen("python3 /home/pi/Documents/Python\ Code/pygame1.py 1", shell=True)

    def draw(self, screen):
        screen.blit(self.rendered, self.pos)

text=("A long time ago, a barbarian strode from the frozen north. Sword in hand...")

message = DynamicText(font, text, (65, 120), autoreset=True)

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT: break
        if event.type == pygame.USEREVENT: message.update()
    else:
        screen.fill(pygame.color.Color('black'))
```

STEP 10

When you run this code it will display a long, narrow Pygame window with the intro text scrolling to the right. After a pause of ten seconds, it then launches the main game Python script where you can move the warrior sprite around. Overall the effect is quite good but there's always room for improvement.



```
A long time ago, a barbarian strode from the frozen north. Sword in hand...
```

Create Your Own Modules

Large programs can be much easier to manage if you break them up into smaller parts and import the parts you need as modules. Learning to build your own modules also makes it easier to understand how they work.

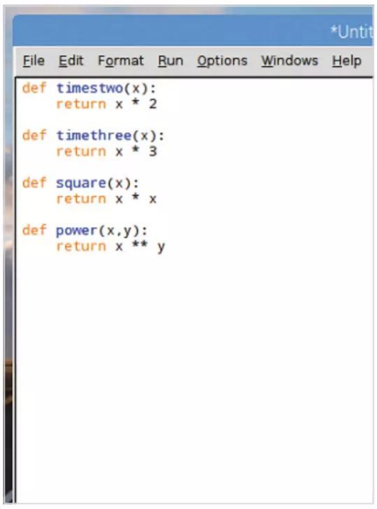
BUILDING MODULES

Modules are Python files, containing code, that you save using a .py extension. These are then imported into Python using the now familiar import command.

STEP 1

Let's start by creating a set of basic mathematics functions. Multiply a number by two, three and square or raise a number to an exponent (power). Create a New File in the IDLE and enter:

```
def timestwo(x):
        return x * 2

def timesthree(x):
        return x * 3

def square(x):
        return x * x

def power(x,y):
        return x ** y
```


*Untit
File Edit Format Run Options Windows Help

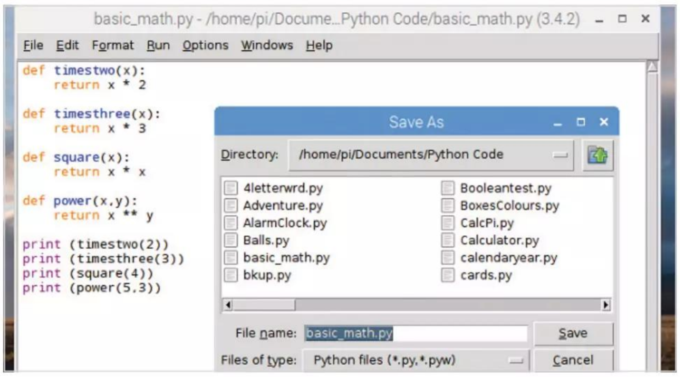
```
def timestwo(x):
    return x * 2

def timethree(x):
    return x * 3

def square(x):
    return x * x

def power(x,y):
    return x ** y
```

STEP 2

Under the above code, enter functions to call the code:

```
print (timestwo(2))
print (timesthree(3))
print (square(4))
print (power(5,3))
```

Save the program as basic_math.py and execute it to get the results.


basic_math.py - /home/pi/Docume...Python Code/basic_math.py (3.4.2)
File Edit Format Run Options Windows Help
Save As
4letterwrd.py
Adventure.py
AlarmClock.py
Balls.py
basic_math.py
bkup.py
Booleantest.py
BoxesColours.py
CalcPi.py
Calculator.py
calendaryear.py
cards.py
Save
Cancel

```
def timestwo(x):
    return x * 2

def timesthree(x):
    return x * 3

def square(x):
    return x * x

def power(x,y):
    return x ** y

print (timestwo(2))
print (timesthree(3))
print (square(4))
print (power(5,3))
```

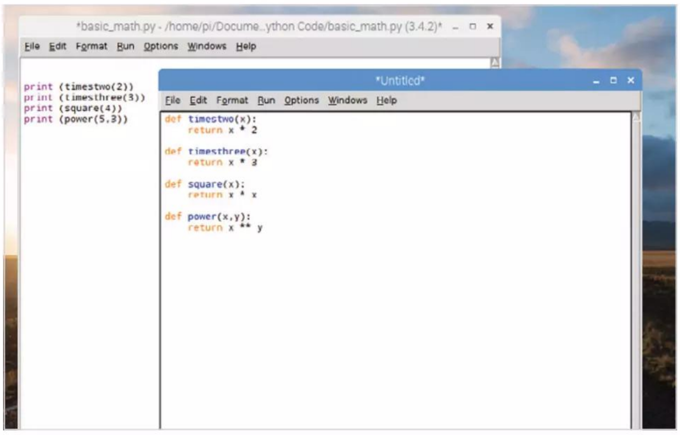
Directory: /home/pi/Documents/Python Code

File name: basic_math.py

Files of type: Python files (*.py,*.pyw)

STEP 3

Now you're going to take the function definitions out of the program and into a separate file. Highlight the function definitions and choose Edit > Cut. Choose File > New File and use Edit > Paste in the new window. You now have two separate files, one with the function definitions, the other with the function calls.

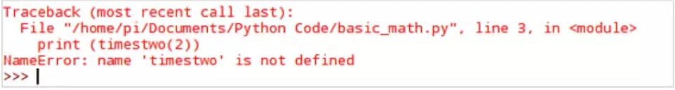

basic_math.py - /home/pi/Docume...ython Code/basic_math.py (3.4.2)
File Edit Format Run Options Windows Help
Untitled
File Edit Format Run Options Windows Help

```
print (timestwo(2))
print (timesthree(3))
print (square(4))
print (power(5,3))
```

```
def timestwo(x):
    return x * 2

def timesthree(x):
    return x * 3

def square(x):
    return x * x

def power(x,y):
    return x ** y
```

STEP 4

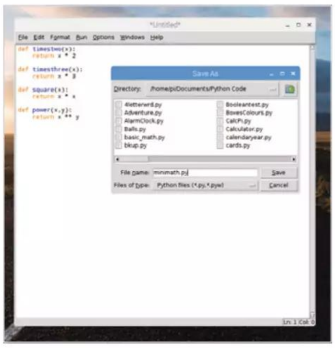
If you now try and execute the basic_math.py code again, the error **'NameError: name 'timestwo' is not defined'** will be displayed. This is due to the code no longer having access to the function definitions.



```
Traceback (most recent call last):
  File "/home/pi/Documents/Python Code/basic_math.py", line 3, in <module>
    print (timestwo(2))
NameError: name 'timestwo' is not defined
>>> |
```

STEP 5

Return to the newly created window containing the function definitions, and click File > Save As. Name this **minimath.py** and save it in the same location as the original **basic_math.py** program. Now close the minimath.py window, so the basic_math.py window is left open.


Untitled
File Edit Format Run Options Windows Help
Save As
4letterwrd.py
Adventure.py
AlarmClock.py
Balls.py
basic_math.py
bkup.py
Booleantest.py
BoxesColours.py
CalcPi.py
Calculator.py
calendaryear.py
cards.py
Save
Cancel
Ln: 1 Col: 0

```
def timestwo(x):
    return x * 2

def timesthree(x):
    return x * 3

def square(x):
    return x * x

def power(x,y):
    return x ** y
```

Directory: /home/pi/Documents/Python Code

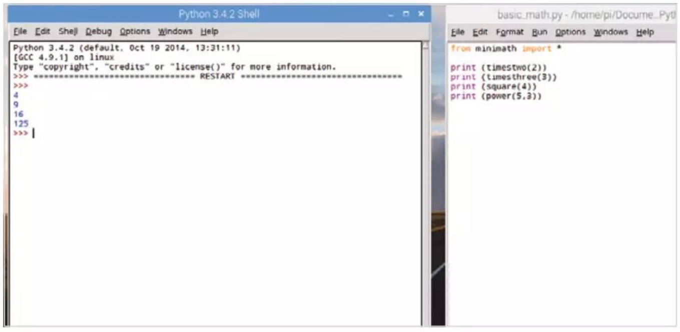
File name: minimath.py

Files of type: Python files (*.py,*.pyw)

STEP 6 Back to the basic_math.py window: at the top of the code enter:

```
from minimath import *
```

This will import the function definitions as a module. Press F5 to save and execute the program to see it in action.

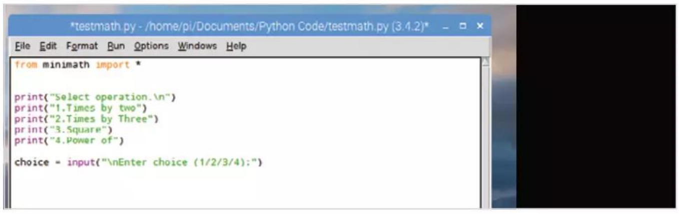

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
basic_math.py - /home/pi/Docume...Pyt
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
4
9
16
125
>>> 
```

```
from minimath import *

print (timestwo(2))
print (timesthree(3))
print (square(4))
print (power(5,3))
```

STEP 7 You can now use the code further to make the program a little more advanced, utilising the newly created module to its full. Include some user interaction. Start by creating a basic menu the user can choose from:

```
print("Select operation.\n")
print("1.Times by two")
print("2.Times by Three")
print("3.Square")
print("4.Power of")

choice = input("\nEnter choice (1/2/3/4):")
```

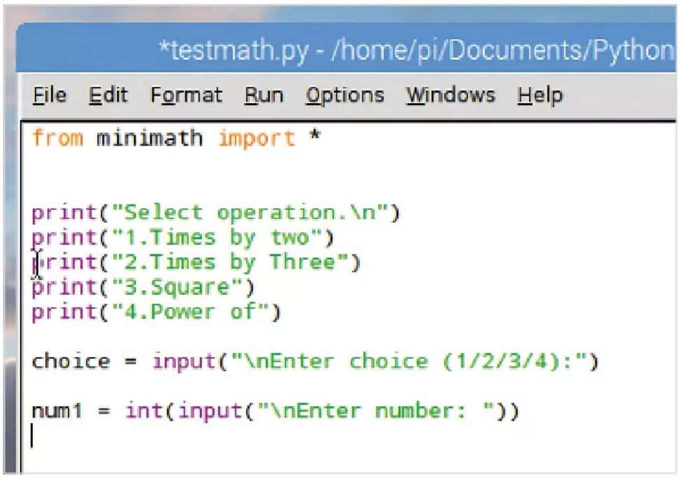

testmath.py - /home/pi/Documents/Python Code/testmath.py (3.4.2)
File Edit Format Run Options Windows Help

```
from minimath import *


print("Select operation.\n")
print("1.Times by two")
print("2.Times by Three")
print("3.Square")
print("4.Power of")

choice = input("\nEnter choice (1/2/3/4):")
```

STEP 8 Now we can add the user input to get the number the code will work on:

```
num1 = int(input("\nEnter number: "))
```

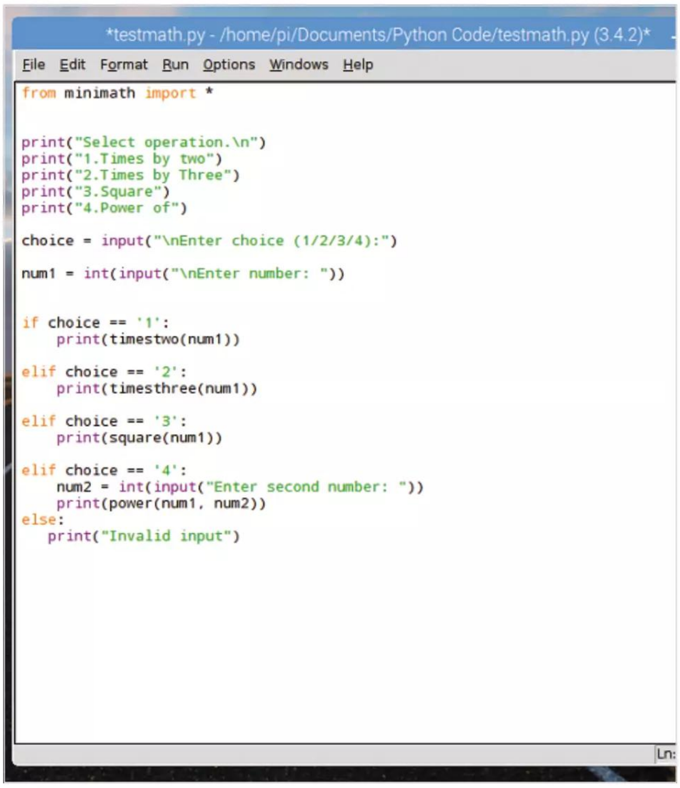
This will save the user-entered number as the variable num1.


*testmath.py - /home/pi/Documents/Python
File Edit Format Run Options Windows Help

```
from minimath import *


print("Select operation.\n")
print("1.Times by two")
print("2.Times by Three")
print("3.Square")
print("4.Power of")

choice = input("\nEnter choice (1/2/3/4):")

num1 = int(input("\nEnter number: "))
```

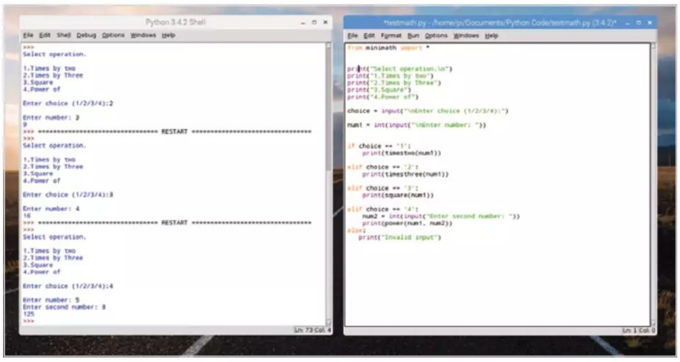
STEP 9 Finally, you can now create a range of if statements to determine what to do with the number and utilise the newly created function definitions:

```
if choice == '1':
    print(timestwo(num1))

elif choice == '2':
    print(timesthree(num1))

elif choice == '3':
    print(square(num1))

elif choice == '4':
    num2 = int(input("Enter second number: "))
    print(power(num1, num2))
else:
   print("Invalid input")
```


testmath.py - /home/pi/Documents/Python Code/testmath.py (3.4.2)
File Edit Format Run Options Windows Help
Ln:

```
from minimath import *


print("Select operation.\n")
print("1.Times by two")
print("2.Times by Three")
print("3.Square")
print("4.Power of")

choice = input("\nEnter choice (1/2/3/4):")

num1 = int(input("\nEnter number: "))


if choice == '1':
    print(timestwo(num1))

elif choice == '2':
    print(timesthree(num1))

elif choice == '3':
    print(square(num1))

elif choice == '4':
    num2 = int(input("Enter second number: "))
    print(power(num1, num2))
else:
   print("Invalid input")
```

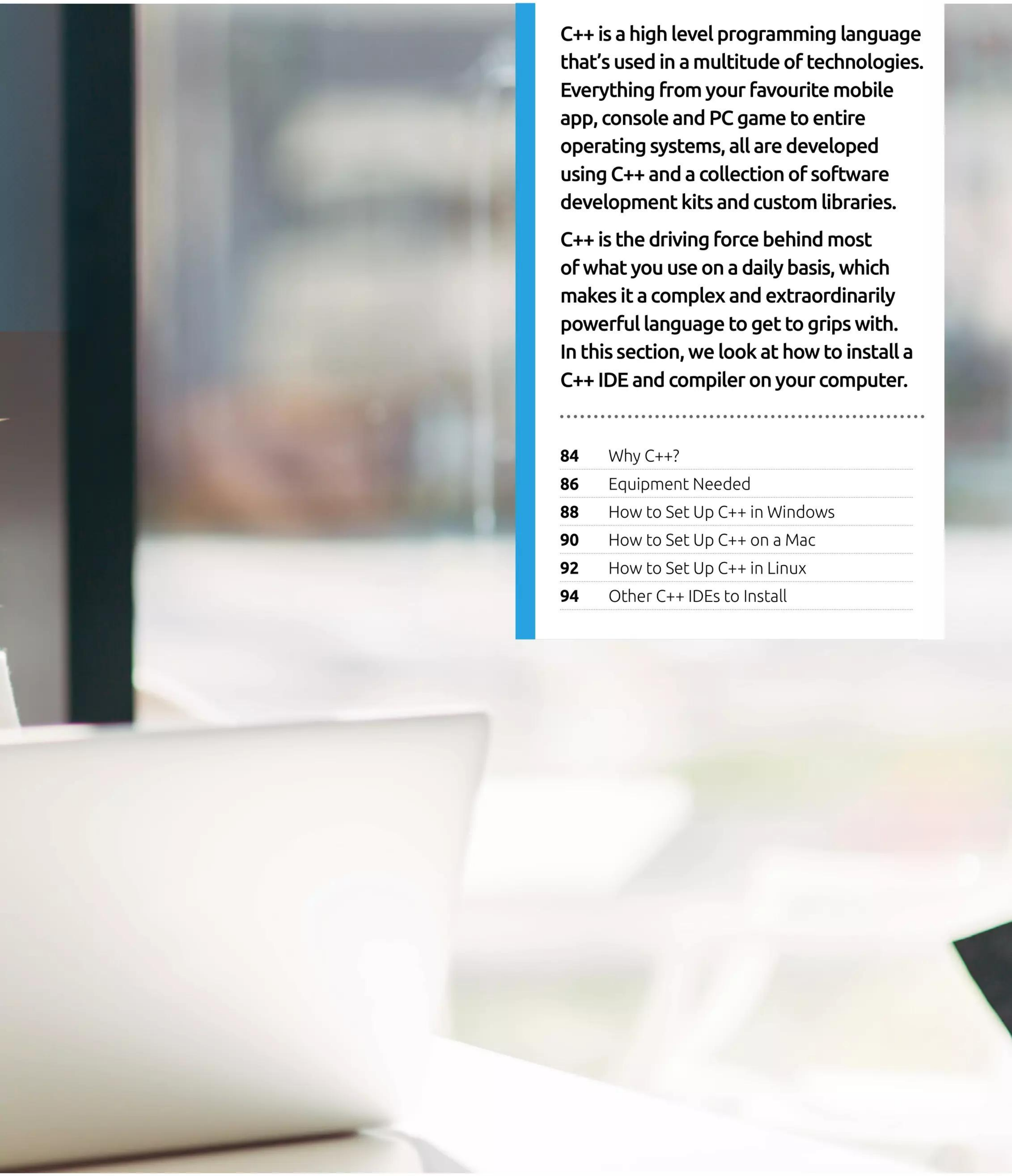
STEP 10 Note that for the last available options, the Power of choice, we've added a second variable, num2. This passes a second number through the function definition called power. Save and execute the program to see it in action.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Ln: 73 Col: 4
testmath.py - /home/pi/Documents/Python Code/testmath.py (3.4.2)
File Edit Format Run Options Windows Help
Ln: 1 Col: 0

```
>>>
Select operation.

1.Times by two
2.Times by Three
3.Square
4.Power of

Enter choice (1/2/3/4):2

Enter number: 3
9
>>> ================================ RESTART ================================
>>>
Select operation.

1.Times by two
2.Times by Three
3.Square
4.Power of

Enter choice (1/2/3/4):3

Enter number: 4
16
>>> ================================ RESTART ================================
>>>
Select operation.

1.Times by two
2.Times by Three
3.Square
4.Power of

Enter choice (1/2/3/4):4

Enter number: 5
Enter second number: 3
125
>>>
```

```
from minimath import *


print("Select operation.\n")
print("1.Times by two")
print("2.Times by Three")
print("3.Square")
print("4.Power of")

choice = input("\nEnter choice (1/2/3/4):")

num1 = int(input("\nEnter number: "))


if choice == '1':
    print(timestwo(num1))

elif choice == '2':
    print(timesthree(num1))

elif choice == '3':
    print(square(num1))

elif choice == '4':
    num2 = int(input("Enter second number: "))
    print(power(num1, num2))
else:
   print("Invalid input")
```

Say Hello to C++



C++ is a high level programming language that's used in a multitude of technologies. Everything from your favourite mobile app, console and PC game to entire operating systems, all are developed using C++ and a collection of software development kits and custom libraries.

C++ is the driving force behind most of what you use on a daily basis, which makes it a complex and extraordinarily powerful language to get to grips with. In this section, we look at how to install a C++ IDE and compiler on your computer.

84 Why C++?
86 Equipment Needed
88 How to Set Up C++ in Windows
90 How to Set Up C++ on a Mac
92 How to Set Up C++ in Linux
94 Other C++ IDEs to Install

Why C++?

C++ is one of the most popular programming languages available today. Originally called C with Classes, the language was renamed C++ in 1983. It's an extension of the original C language and is a general purpose object-oriented (OOP) environment.

C EVERYTHING

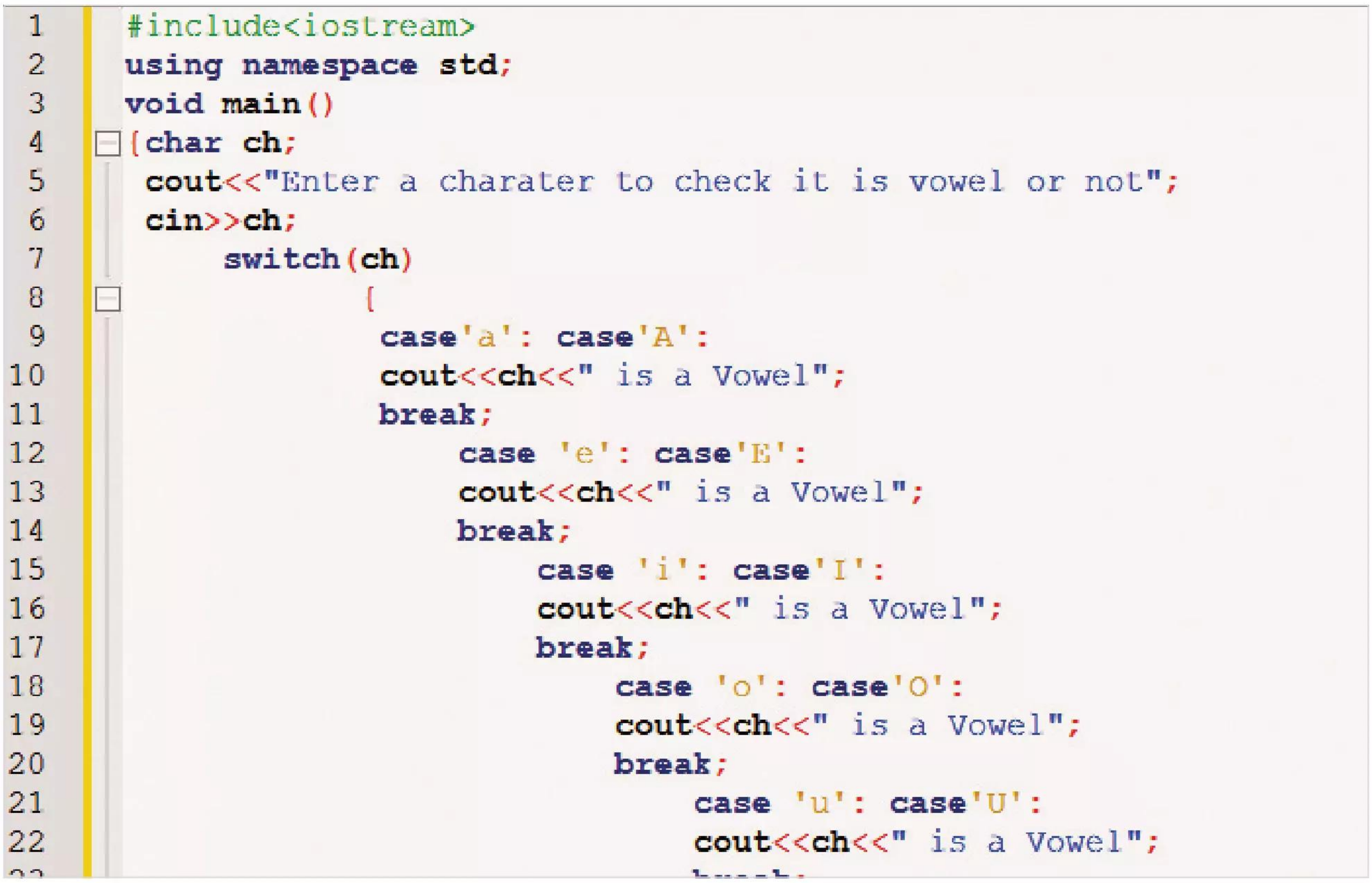
Due to how complex the language can be, and its power and performance, C++ is often used to develop games, programs, device drivers and even entire operating systems.

Dating back to 1979, the start of the golden era of home computing, C++, or rather C with Classes, was the brainchild of Danish computer scientist Bjarne Stroustrup while working on his PhD thesis. Stroustrup's plan was to further the original C language, which was widely used since the early seventies.

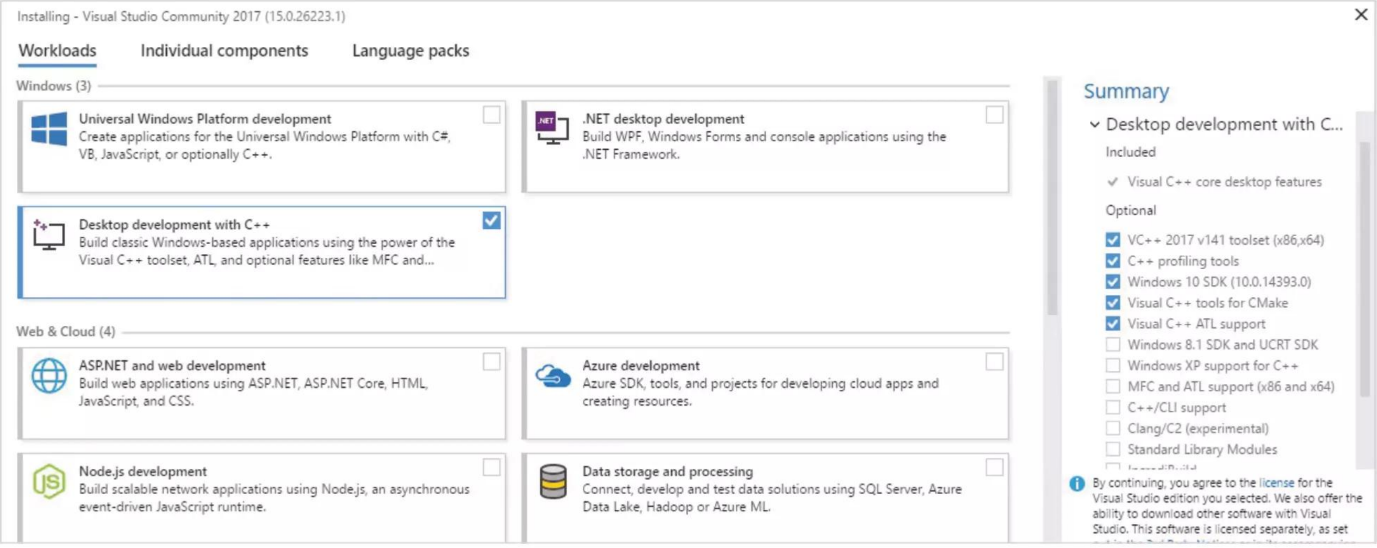
C++ proved to be popular among the developers of the '80s, since it was a much easier environment to get to grips with and more importantly, it was 99% compatible with the original C language. This meant that it could be used beyond the mainstream computing labs and by regular people who didn't have access to the mainframes and large computing data centres.

C++'s impact in the digital world is immense. Many of the programs, applications, games and even operating systems are coded using C++. For example, all of Adobe's major applications, such as Photoshop, InDesign and so on, are developed in C++. You will find that the browser you surf the Internet with is written in C++, as well as Windows 10, Microsoft Office and the backbone to Google's search engine. Apple's macOS is written largely in C++ (with some

C++ code is much faster than that of Python.



```
#include<iostream>
using namespace std;
void main()
{char ch;
 cout<<"Enter a charater to check it is vowel or not";
 cin>>ch;
     switch(ch)
            {
             case'a': case'A':
             cout<<ch<<" is a Vowel";
             break;
                 case 'e': case'E':
                 cout<<ch<<" is a Vowel";
                 break;
                     case 'i': case'I':
                     cout<<ch<<" is a Vowel";
                     break;
                         case 'o': case'O':
                         cout<<ch<<" is a Vowel";
                         break;
                             case 'u': case'U':
                             cout<<ch<<" is a Vowel";
```



Installing - Visual Studio Community 2017 (15.0.26223.1)

Workloads | Individual components | Language packs

Windows (3)

☐ **Universal Windows Platform development**
Create applications for the Universal Windows Platform with C#, VB, JavaScript, or optionally C++.

☐ **.NET desktop development**
Build WPF, Windows Forms and console applications using the .NET Framework.

☑ **Desktop development with C++**
Build classic Windows-based applications using the power of the Visual C++ toolset, ATL, and optional features like MFC and...

Web & Cloud (4)

☐ **ASP.NET and web development**
Build web applications using ASP.NET, ASP.NET Core, HTML, JavaScript, and CSS.

☐ **Azure development**
Azure SDK, tools, and projects for developing cloud apps and creating resources.

☐ **Node.js development**
Build scalable network applications using Node.js, an asynchronous event-driven JavaScript runtime.

☐ **Data storage and processing**
Connect, develop and test data solutions using SQL Server, Azure Data Lake, Hadoop or Azure ML.

Summary

Desktop development with C...

Included

✓ Visual C++ core desktop features

Optional

☑ VC++ 2017 v141 toolset (x86,x64)
☑ C++ profiling tools
☑ Windows 10 SDK (10.0.14393.0)
☑ Visual C++ tools for CMake
☑ Visual C++ ATL support
☐ Windows 8.1 SDK and UCRT SDK
☐ Windows XP support for C++
☐ MFC and ATL support (x86 and x64)
☐ C++/CLI support
☐ Clang/C2 (experimental)
☐ Standard Library Modules

By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set

Microsoft's Visual Studio is a great, free environment to learn C++ in.

other languages mixed in depending on the function) and the likes of NASA, SpaceX and even CERN use C++ for various applications, programs, controls and umpteen other computing tasks.

C++ is also extremely efficient and performs well across the board as well as being an easier addition to the core C language. This higher level of performance over other languages, such as Python, BASIC and such, makes it an ideal development environment for modern computing, hence the aforementioned companies using it so widely.

While Python is a great programming language to learn, C++ puts the developer in a much wider world of coding. By mastering C++, you can find yourself developing code for the likes of Microsoft, Apple and so on. Generally, C++ developers enjoy a higher salary than programmers of some other languages and due to its versatility, the C++ programmer can move between jobs and companies without the need to relearn anything specific. However, Python is an easier language to begin with. If you're completely new to programming then we would recommend you begin with Python and spend some time getting to grips with programming structure and the many ways and means in which you find a solution to a problem through programming. Once you can happily power up your computer and whip out a Python program with one hand tied behind your back, then move on to C++. Of course, there's nothing stopping you from jumping straight into C++; if you feel up to the task, go for it.

Getting to use C++ is as easy as Python, all you need is the right set of tools in which to communicate with the computer in C++ and you can start your journey. A C++ IDE is free of charge, even the immensely powerful Visual Studio from Microsoft is freely available to download and use. You can get into C++ from any operating system, be it macOS, Linux, Windows or even mobile platforms.

Just like Python, to answer the question of Why C++ is the answer is because it's fast, efficient and developed by most of the applications you regularly use. It's cutting edge and a fantastic language to master.

Microsoft Windows
Mac
debian
ubuntu

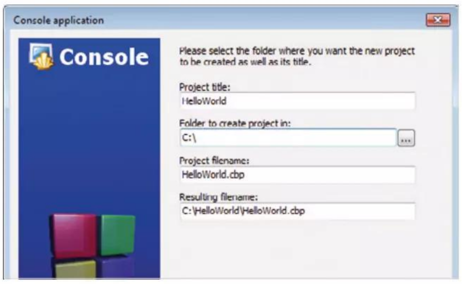
Indeed, the operating system you're using is written in C++.

Equipment Needed

You don't need to invest a huge amount of money in order to learn C++ and you don't need an entire computing lab at your disposal either. Providing you have a fairly modern computer, everything else is freely available.

C++ SETUPS

Most, if not all, operating systems have C++ in their code, so it stands to reason that you can learn to program in C++ no matter what OS you're currently using.



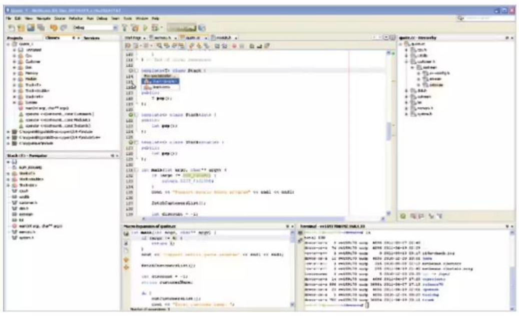
Please select the folder where you want the new project to be created as well as its title.

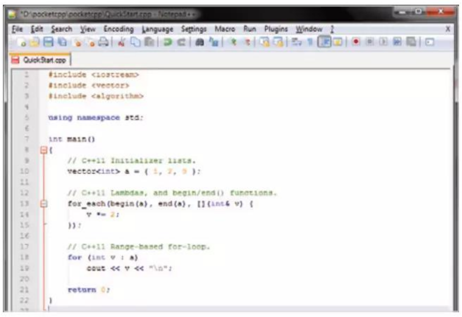
Project title: HelloWorld

Folder to create project in: C:\

Project filename: HelloWorld.cbp

Resulting filename: C:\HelloWorld\HelloWorld.cbp




*D:\pocketcpp\pocketcpp\QuickStart.cpp - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
QuickStart.cpp

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main()
{
    // C++11 Initializer lists.
    vector<int> a = { 1, 2, 3 };

    // C++11 Lambdas, and begin/end() functions.
    for_each(begin(a), end(a), [](int& v) {
        v *= 2;
    });

    // C++11 Range-based for-loop.
    for (int v : a)
        cout << v << "\n";

    return 0;
}
```

☐ COMPUTER

Unless you fancy writing out your C++ code by hand on a sheet of paper (which is something many older coders used to do), a computer is an absolute must have component. PC users can have any recent Linux distro or Windows OS, Mac users the latest macOS.

☐ AN IDE

Just as with Python, an IDE is used to enter and execute your C++ code. Many IDEs come with extensions and plugins that help make it work better, or add an extra level of functionality. Often, an IDE provides enhancements depending on the core OS being used, such as being enhanced for Windows 10.

☐ COMPILER

A compiler is a program that converts the C++ language into binary, so that the computer can understand. While some IDEs come with a compiler built in, others don't. Code::Blocks is our favourite IDE that comes with a C++ compiler as part of the package. More on this later.

☐ TEXT EDITOR

Some programmers much prefer to use a text editor to assemble their C++ code before running it through a compiler. Essentially you can use any text editor to write code, just save it with a .cpp extension. However, Notepad++ is one of the best code text editors available.

☐ INTERNET ACCESS

While it's entirely possible to learn how to code on a computer that's not attached to the Internet, it's extraordinarily difficult. You need to install relevant software, keep it up to date, install any extras or extensions and look for help when coding. All of these require access to the Internet.

☐ TIME AND PATIENCE

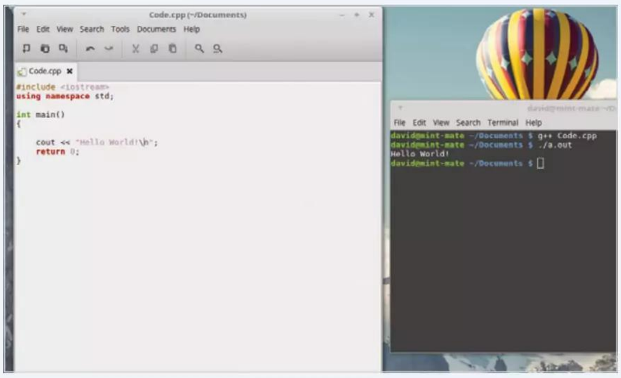
Yes, as with Python, you're going to need to set aside significant time to spend on learning how to code in C++. Sadly, unless you're a genius, it's not going to happen overnight, or even a week. A good C++ coder has spent many years honing their craft, so be patient, start small and keep learning.

OS SPECIFIC NEEDS

C++ will work in any operating system but getting all the necessary pieces together can be confusing to a newcomer. Here are some OS specifics for C++.

LINUX

Linux users are lucky in that they already have a compiler and text editor built into their operating system. Any text editor allows you to type out your C++ code, when it's saved with a .cpp extension, use g++ to compile it.

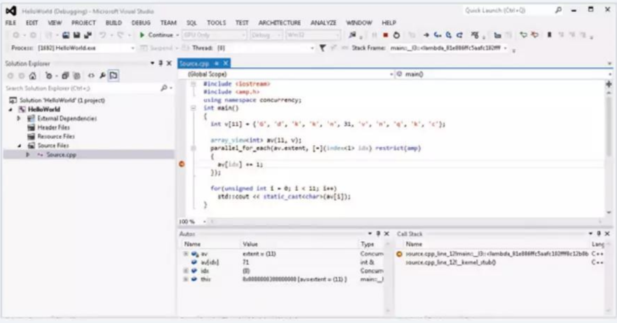

Code.cpp (~/Documents)
File Edit View Search Tools Documents Help
Code.cpp
File Edit View Search Terminal Help

```
#include <iostream>
using namespace std;

int main()
{

    cout << "Hello World!\n";
    return 0;
}
```

```
david@mint-mate ~/Documents $ g++ Code.cpp
david@mint-mate ~/Documents $ ./a.out
Hello World!
david@mint-mate ~/Documents $
```

WINDOWS

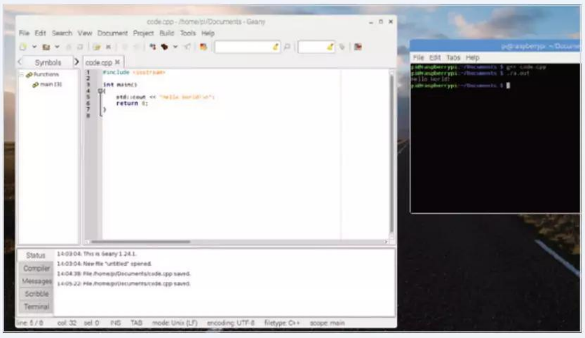
We have mentioned previously that a good IDE is Microsoft's Visual Studio. However, a better IDE and compiler is Code::Blocks, which is regularly kept up to date with a new release twice a year. Otherwise Windows users can enter their code in Notepad++, then compile it with MinGW as used by Code::Blocks.


HelloWorld (Debugging) - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL TOOLS TEST ARCHITECTURE ANALYZE WINDOW HELP
Quick Launch (Ctrl+Q)
Continue
Thread: [0]
Stack Frame:
Solution Explorer
Solution 'HelloWorld' (1 project)
HelloWorld
External Dependencies
Header Files
Resource Files
Source Files
Source.cpp
(Global Scope)
main()
100 %
Autos
Call Stack

```
#include <iostream>
#include <amp.h>
using namespace concurrency;
int main()
{
  int v[11] = {'G', 'd', 'k', 'k', 'n', 31, 'v', 'n', 'q', 'k', 'c'};

  array_view<int> av(11, v);
  parallel_for_each(av.extent, [=](index<1> idx) restrict(amp)
  {
    av[idx] += 1;
  });

  for(unsigned int i = 0; i < 11; i++)
    std::cout << static_cast<char>(av[i]);
}
```

RASPBERRY PI

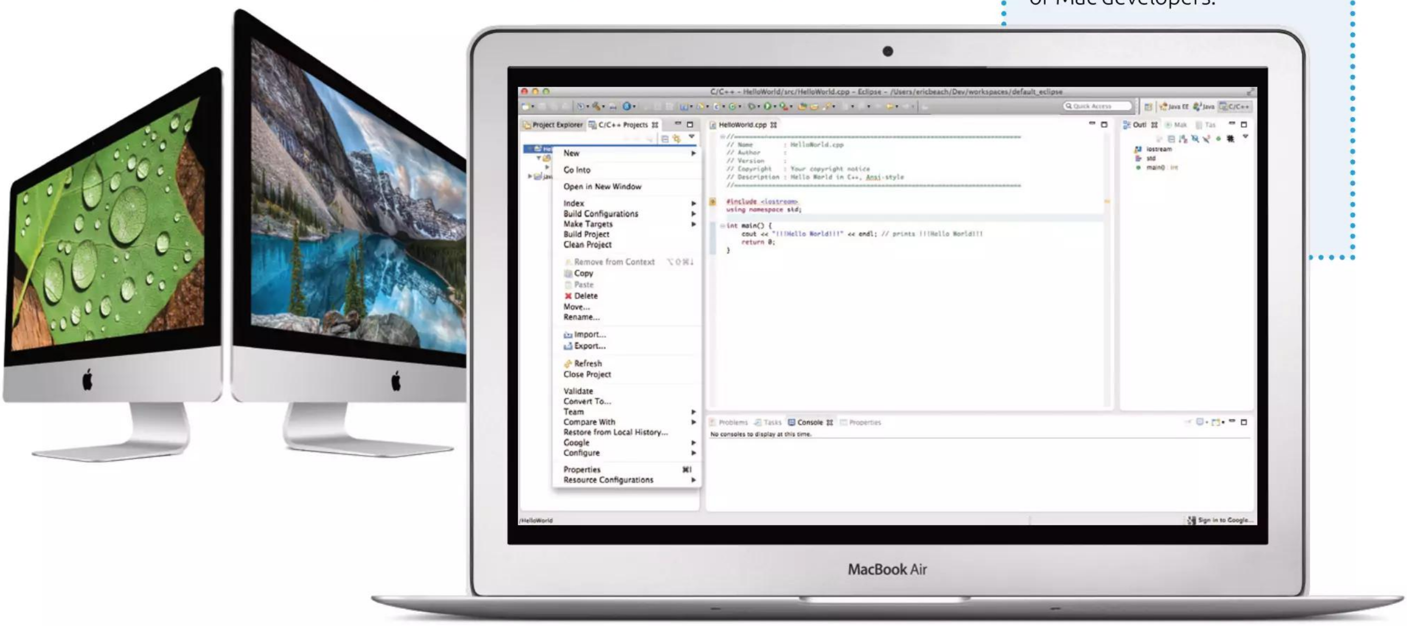
The Raspberry Pi's operating system is Raspbian, which is Linux based. Therefore, you're able to write your code out using a text editor, then compile it with g++ as you would in any other Linux distro.


code.cpp - /home/pi/Documents - Geany
File Edit Search View Document Project Build Tools Help
Symbols
code.cpp
Functions
main [3]
Status
Compiler
Messages
Scribble
Terminal
File Edit Tabs Help

```
14:03:04: This is Geany 1.24.1.
14:03:04: New file "untitled" opened.
14:04:38: File /home/pi/Documents/code.cpp saved.
14:05:22: File /home/pi/Documents/code.cpp saved.
```

MAC

Mac owners will need to download and install Xcode to be able to compile their C++ code natively. Other options for the macOS include Netbeans, Eclipse or Code::Blocks. Note: the latest Code::Blocks isn't available for Mac due to a lack of Mac developers.


C/C++ - HelloWorld/src/HelloWorld.cpp - Eclipse - /Users/ericbeach/Dev/workspaces/default_eclipse
Project Explorer
C/C++ Projects
HelloWorld.cpp
New
Go Into
Open in New Window
Index
Build Configurations
Make Targets
Build Project
Clean Project
Remove from Context
Copy
Paste
Delete
Move...
Rename...
Import...
Export...
Refresh
Close Project
Validate
Convert To...
Team
Compare With
Restore from Local History...
Google
Configure
Properties
Resource Configurations
Problems
Tasks
Console
No consoles to display at this time.
MacBook Air

```
//============================================================================
// Name        : HelloWorld.cpp
// Author      :
// Version     :
// Copyright   : Your copyright notice
// Description : Hello World in C++, Ansi-style
//============================================================================

#include <iostream>
using namespace std;

int main() {
	cout << "!!!Hello World!!!" << endl; // prints !!!Hello World!!!
	return 0;
}
```

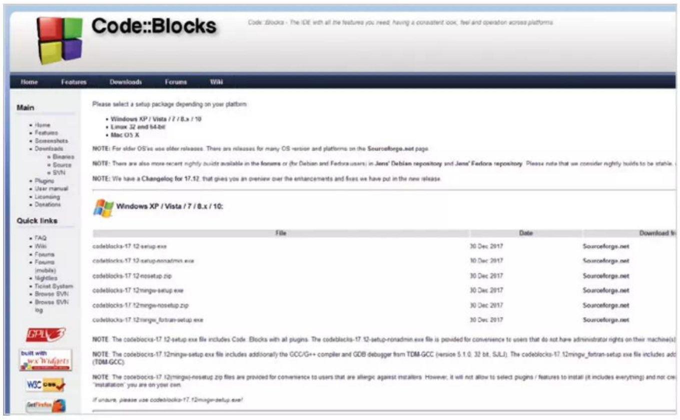
How to Set Up C++ in Windows

Windows users have a wealth of choice when it comes to programming in C++. There are plenty of IDEs and compilers available, including Visual Studio from Microsoft. However, in our opinion, the best C++ IDE to begin with is Code::Blocks.

CODE::BLOCKS

Code::Blocks is a free C++, C and Fortran IDE that's feature rich and easily extendible with plug-ins. It's easy to use, comes with a compiler and has a vibrant community behind it.

STEP 1 Start by visiting the Code::Blocks download site, at **www.codeblocks.org/downloads**. From there, click on the 'Download the binary releases' link to be taken to the latest downloadable version for Windows.



Code::Blocks

Home Features Downloads Forums Wiki

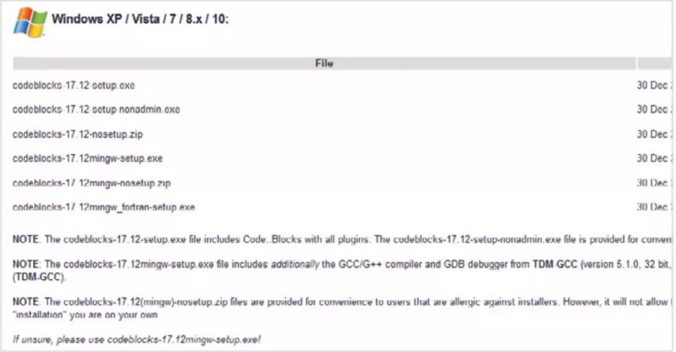
Please select a setup package depending on your platform:

- Windows XP / Vista / 7 / 8.x / 10
- Linux 32 and 64-bit
- Mac OS X

Windows XP / Vista / 7 / 8.x / 10:

File	Date	Download fr
codeblocks-17.12-setup.exe	30 Dec 2017	Sourceforge.net
codeblocks-17.12-setup-nonadmin.exe	30 Dec 2017	Sourceforge.net
codeblocks-17.12-nosetup.zip	30 Dec 2017	Sourceforge.net
codeblocks-17.12mingw-setup.exe	30 Dec 2017	Sourceforge.net
codeblocks-17.12mingw-nosetup.zip	30 Dec 2017	Sourceforge.net
codeblocks-17.12mingw_fortran-setup.exe	30 Dec 2017	Sourceforge.net

STEP 2 You can see that there are several Windows versions available. The one you want to download has 'mingw-setup.exe' at the end of the current version number. At the time of writing this is: codeblocks-17.12mingw-setup.exe. The difference is that the mingw-setup version includes a C++ compiler and debugger from TDM-GCC (a compiler suite).



Windows XP / Vista / 7 / 8.x / 10:

File	
codeblocks-17.12-setup.exe	30 Dec
codeblocks-17.12-setup-nonadmin.exe	30 Dec
codeblocks-17.12-nosetup.zip	30 Dec
codeblocks-17.12mingw-setup.exe	30 Dec
codeblocks-17.12mingw-nosetup.zip	30 Dec
codeblocks-17.12mingw_fortran-setup.exe	30 Dec

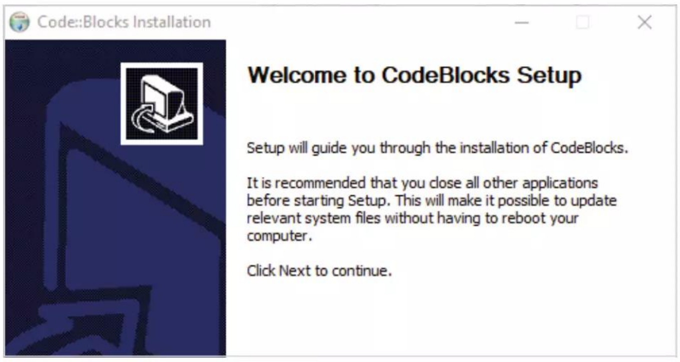
NOTE: The codeblocks-17.12-setup.exe file includes Code::Blocks with all plugins. The codeblocks-17.12-setup-nonadmin.exe file is provided for conven

NOTE: The codeblocks-17.12mingw-setup.exe file includes *additionally* the GCC/G++ compiler and GDB debugger from TDM-GCC (version 5.1.0, 32 bit, (TDM-GCC).

NOTE: The codeblocks-17.12(mingw)-nosetup.zip files are provided for convenience to users that are allergic against installers. However, it will not allow "installation" you are on your own

If unsure, please use codeblocks-17.12mingw-setup.exe!

STEP 3 When you've located the file, click on the Sourceforge.net link at the end of the line and a download notification window appears; click on Save File to start the download and save the executable to your PC. Locate the downloaded Code::Blocks installer and double-click to start. Follow the on-screen instructions to begin the installation.



Code::Blocks Installation

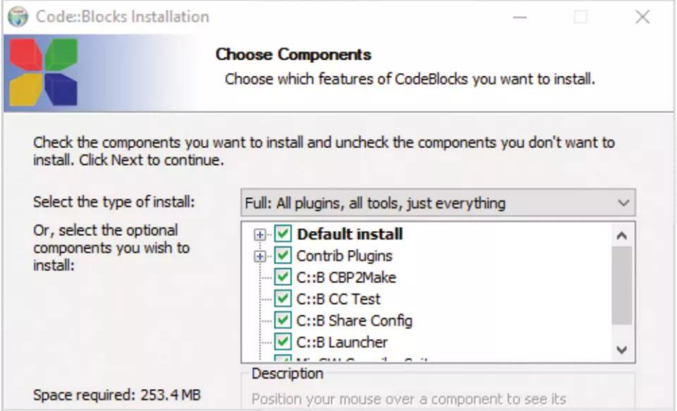
Welcome to CodeBlocks Setup

Setup will guide you through the installation of CodeBlocks.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

STEP 4 Once you've agreed to the licencing terms, there is a choice of installation options available. You can opt for a smaller install, missing out on some of the components but we would recommend you opt for the Full option as default.



Code::Blocks Installation

Choose Components

Choose which features of CodeBlocks you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select the type of install: Full: All plugins, all tools, just everything

Or, select the optional components you wish to install:

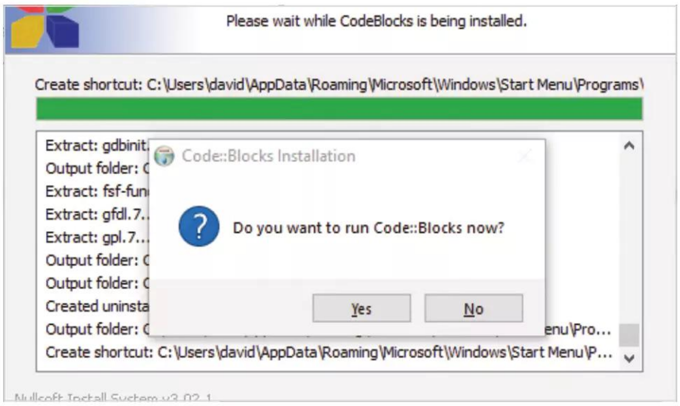
- ☑ **Default install**
- ☑ Contrib Plugins
- ☑ C::B CBP2Make
- ☑ C::B CC Test
- ☑ C::B Share Config
- ☑ C::B Launcher

Space required: 253.4 MB

Description

Position your mouse over a component to see its

STEP 5 Next choose an install location for the Code::Blocks files. It's your choice, but the default will generally suffice, unless of course you have any special requirements. When you click Next, the install begins; when it's finished a notification pops up asking you if you want to start Code::Blocks now, so click Yes.


Please wait while CodeBlocks is being installed.

Create shortcut: C:\Users\david\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\

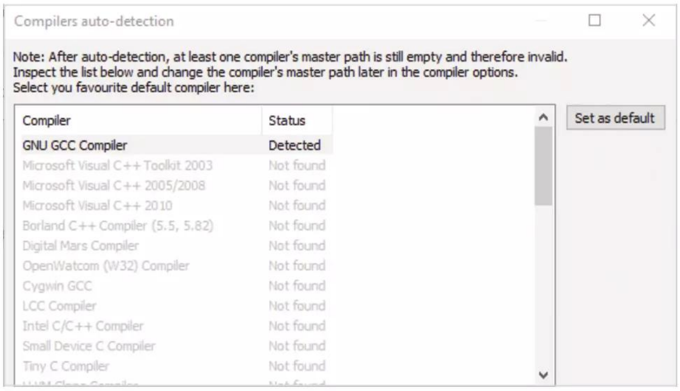
```
Extract: gdbinit.
Output folder: C
Extract: fsf-fun
Extract: gfdl.7..
Extract: gpl.7...
Output folder: C
Output folder: C
Created uninsta
Output folder: C                                                enu\Pro...
Create shortcut: C:\Users\david\AppData\Roaming\Microsoft\Windows\Start Menu\P...
```

Code::Blocks Installation

Do you want to run Code::Blocks now?

Yes
No
Nullsoft Install System v3.02.1

STEP 6 The first time Code::Blocks loads it runs an auto-detect for any C++ compilers you may already have installed on you system. If you don't have any, click on the first detected option, GNU GCC Compiler, and click the Default button to set it as the system's C++ compiler. Click OK when you're ready to continue.

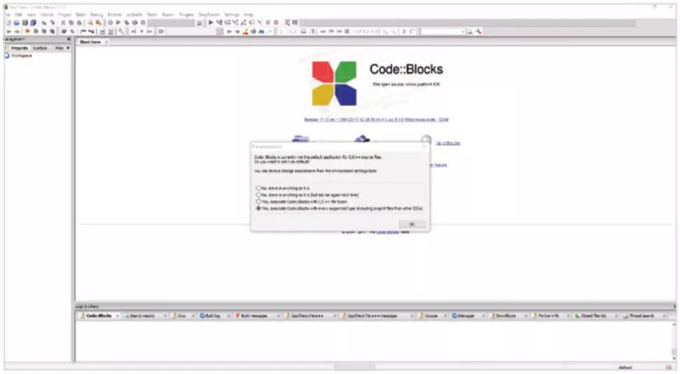

Compilers auto-detection

Note: After auto-detection, at least one compiler's master path is still empty and therefore invalid. Inspect the list below and change the compiler's master path later in the compiler options.
Select you favourite default compiler here:

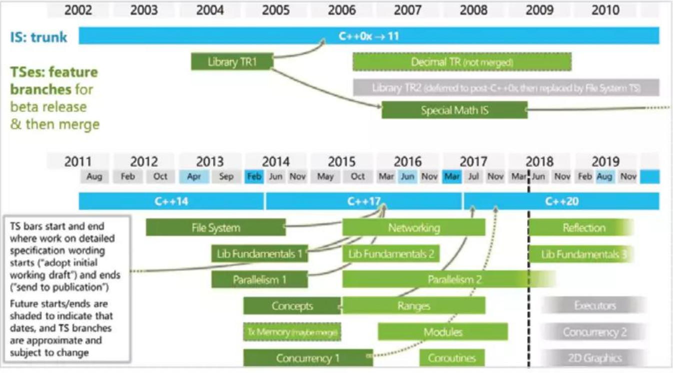
Compiler	Status
GNU GCC Compiler	Detected
Microsoft Visual C++ Toolkit 2003	Not found
Microsoft Visual C++ 2005/2008	Not found
Microsoft Visual C++ 2010	Not found
Borland C++ Compiler (5.5, 5.82)	Not found
Digital Mars Compiler	Not found
OpenWatcom (W32) Compiler	Not found
Cygwin GCC	Not found
LCC Compiler	Not found
Intel C/C++ Compiler	Not found
Small Device C Compiler	Not found
Tiny C Compiler	Not found

Set as default

STEP 7 When the program starts another message appears, informing you that Code::Blocks is currently not the default application for C++ files. You a couple of options: to leave everything as it is or allow Code::Blocks to associate all C++ file types. Again, we would recommend you opt for the last choice to associate Code::Blocks with every supported file type.

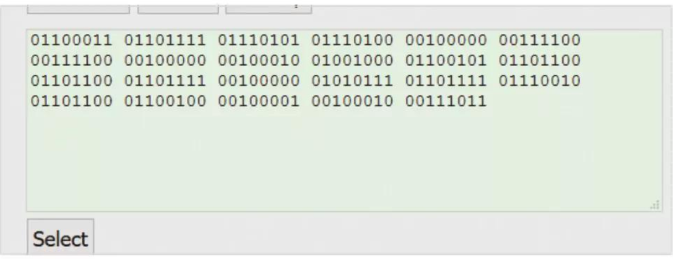

Code::Blocks

STEP 8 Before you start using Code::Blocks it's worth explaining exactly why you need the added compiler. First, a compiler is a separate program that reads through your C++ code and checks it against the latest acceptable programming standards; this is why you need the most recently available compiler. This is currently C++17, with C++20 underway.


2002
2003
2004
2005
2006
2007
2008
2009
2010
IS: trunk
C++0x → 11
TSes: feature branches for beta release & then merge
Library TR1
Decimal TR (not merged)
Library TR2 (deferred to post-C++0x, then replaced by File System TS)
Special Math IS
2011
2012
2013
2014
2015
2016
2017
2018
2019
Aug
Feb
Oct
Apr
Sep
Feb
Jun
Nov
May
Oct
Mar
Jun
Nov
Mar
Jul
Nov
Mar
Jun
Nov
Feb
Aug
Nov
C++14
C++17
C++20
TS bars start and end where work on detailed specification wording starts ("adopt initial working draft") and ends ("send to publication")
Future starts/ends are shaded to indicate that dates, and TS branches are approximate and subject to change
File System
Networking
Reflection
Lib Fundamentals 1
Lib Fundamentals 2
Lib Fundamentals 3
Parallelism 1
Parallelism 2
Concepts
Ranges
Executors
Tx Memory (maybe merge)
Modules
Concurrency 2
Concurrency 1
Coroutines
2D Graphics

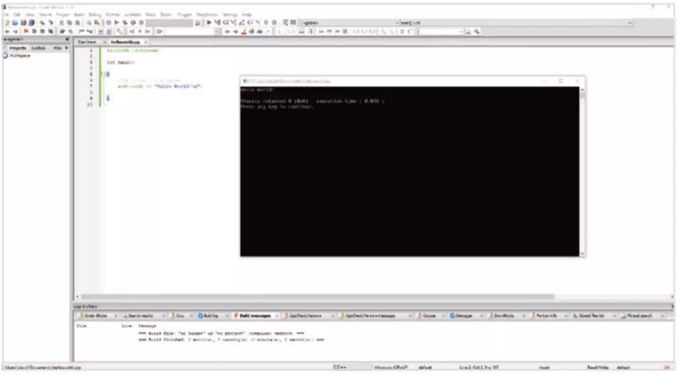
STEP 9 Essentially, computers work and understand only binary, ones and zeros, or Machine Language. Programming in binary isn't effective for human beings. For example, to output the words "Hello World!" to the screen in C++ would appear in binary as:

```
01100011 01101111 01110101 01110100 00100000
00111100 00111100 00100000 00100010 01001000
01100101 01101100 01101100 01101111 00100000
01010111 01101111 01110010 01101100 01100100
00100001 00100010 00111011
```



```
01100011 01101111 01110101 01110100 00100000 00111100
00111100 00100000 00100010 01001000 01100101 01101100
01101100 01101111 00100000 01010111 01101111 01110010
01101100 01100100 00100001 00100010 00111011
```

STEP 10 The compiler therefore takes what you've entered as C++ code and translates that to Machine Language. To execute C++ code the IDE 'builds' the code, checking for errors, then pass it through the compiler to check standardisation and convert it to ones and zeros for the computer to act upon. It's rather clever stuff, when you stop to think about it.



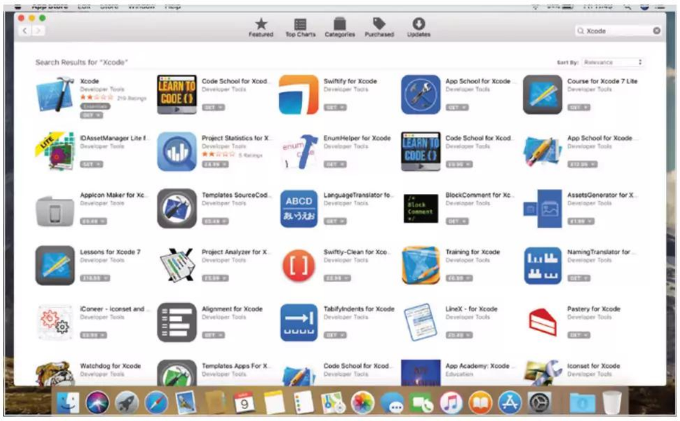
How to Set Up C++ on a Mac

To begin C++ coding on a Mac you first need to install Apple's Xcode. This is a free, full featured IDE that's designed to create native Apple apps. However, you can also use it to create C++ code relatively easily.

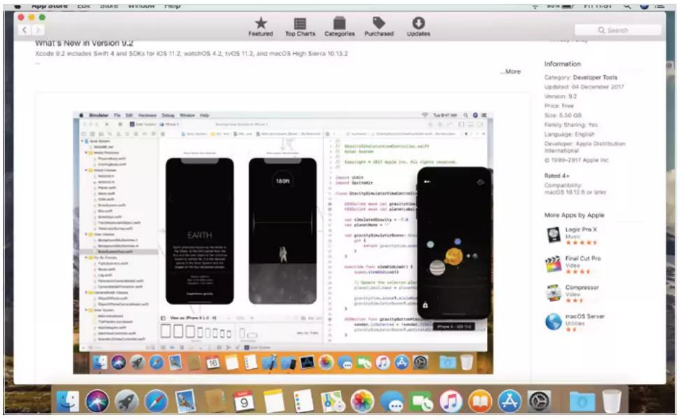
XCODE

Apple's Xcode is primarily designed for users to develop apps for macOS, iOS, tvOS and watchOS applications in Swift or Objective-C but you can use it for C++ too.

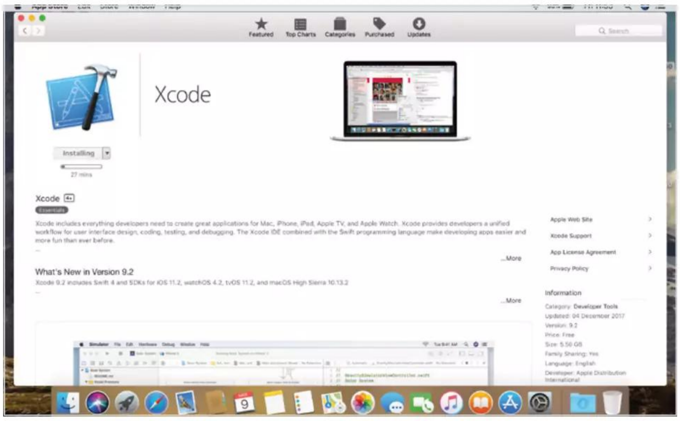
STEP 1 Start by opening the App Store on your Mac, Apple Menu > App Store. In the Search box enter 'Xcode' and press Return. There are many suggestions filling the App Store window but it's the first option, Xcode, that you need to click on.


Search Results for "Xcode"
Xcode

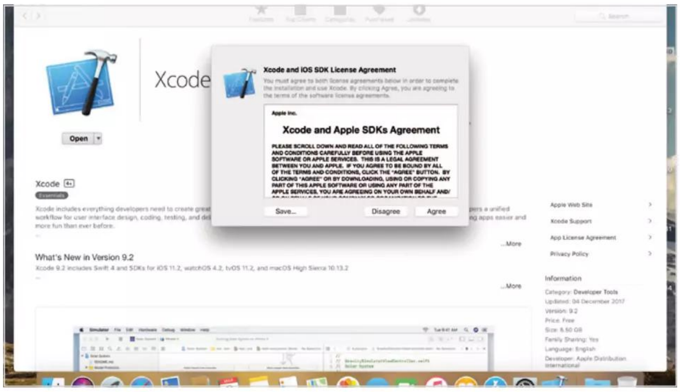
STEP 2 Take a moment to browse through the app's information, including the compatibility, to ensure you have the correct version of macOS. Xcode requires macOS 10.12.6 or later to install and work.


What's New in Version 9.2
Information

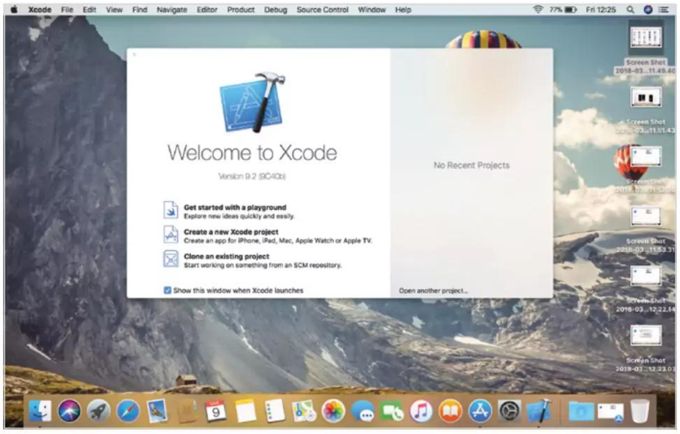
STEP 3 When you're ready, click on the Get button which then turns into Install App. Enter your Apple ID and Xcode begins to download and install. It may take some time depending on the speed of your Internet connection.


Xcode
Installing
What's New in Version 9.2

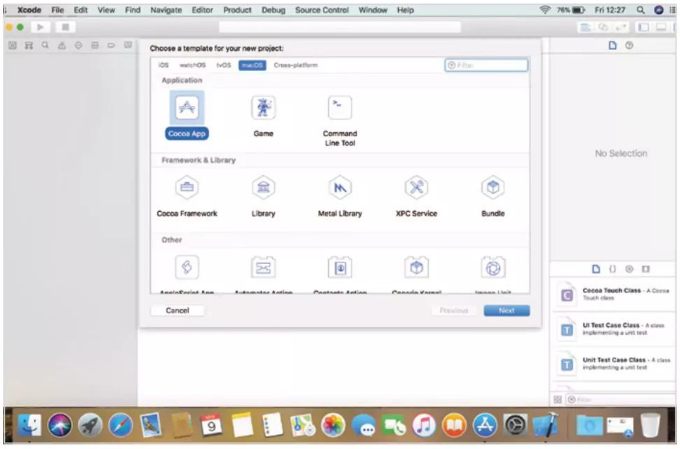
STEP 4 When the installation is complete, click on the Open button to launch Xcode. Click Agree to the licence terms and enter your password to allow Xcode to make changes to the system. When that is done, Xcode begins to install additional components.


Xcode
Open
Xcode and iOS SDK License Agreement
Xcode and Apple SDKs Agreement
Save...
Disagree
Agree

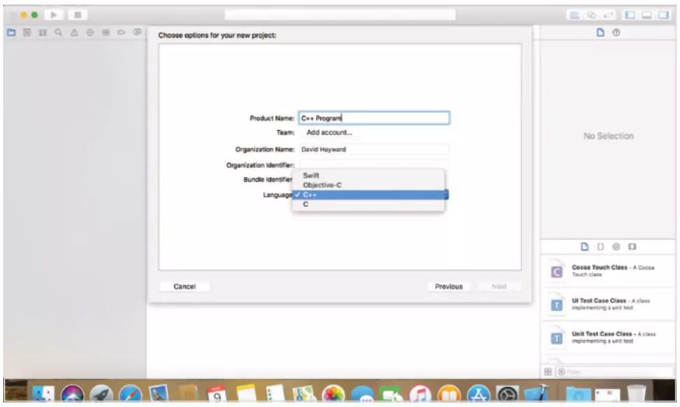
STEP 5 With everything now installed, including the additional components, Xcode launches, displaying the version number along with three choices and any recent projects that you've worked on; with a fresh install though, this is blank.


Welcome to Xcode
Version 9.2 (9C40b)
Get started with a playground
Explore new ideas quickly and easily.
Create a new Xcode project
Create an app for iPhone, iPad, Mac, Apple Watch or Apple TV.
Clone an existing project
Start working on something from an SCM repository.
Show this window when Xcode launches
No Recent Projects
Open another project...

STEP 6 Start by clicking on Create New Xcode Project; this opens a template window to choose which platform you're developing code for. Click the macOS tab, then click the Command Line Tool option. Click Next to continue.


Choose a template for your new project:
iOS
watchOS
tvOS
macOS
Cross-platform
Application
Cocoa App
Game
Command Line Tool
Framework & Library
Cocoa Framework
Library
Metal Library
XPC Service
Bundle
Other
Cancel
Previous
Next
No Selection

STEP 7 Fill in the various fields but ensure that the Language option at the bottom is set to C++; simply choose it from the drop-down list. When you've filled in the fields, and made sure that C++ is the chosen language, click on the Next button to continue.

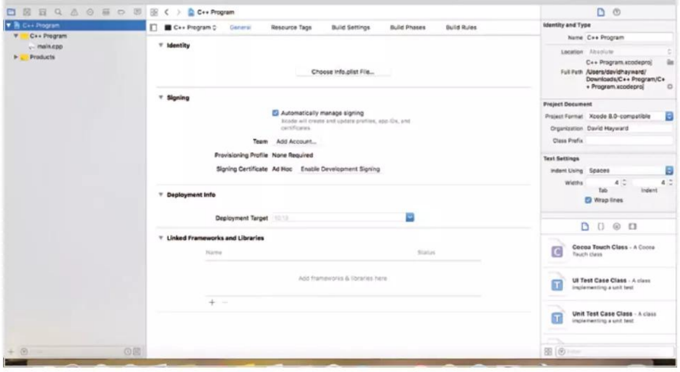


Choose options for your new project:

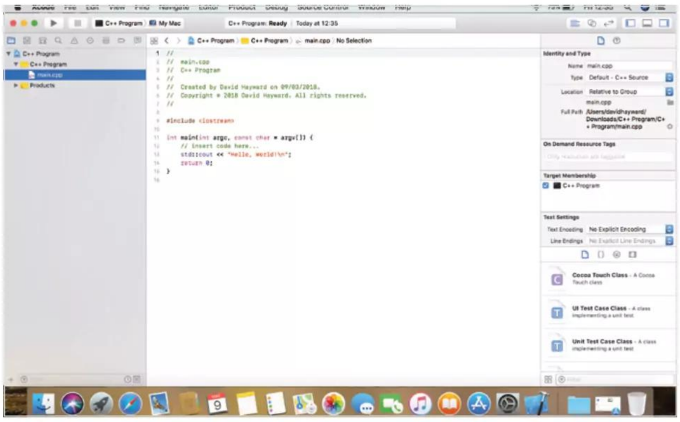
Product Name: C++ Program
Team: Add account...
Organization Name: David Hayward
Language: C++

Organization Identifier
Bundle Identifier
Swift
Objective-C
C++
C
Cancel
Previous
Next
No Selection

STEP 8 The next step asks where to create a Git Repository for all your future code. Choose a location on your Mac, or a network location, and click the Create button. When you've done all that, you can start to code. The left-hand pane details the files used in the C++ program you're coding. Click on the main.cpp file in the list.


C++ Program
main.cpp
Products
General
Resource Tags
Build Settings
Build Phases
Build Rules
Identity
Signing
Automatically manage signing
Deployment Info
Linked Frameworks and Libraries
Identity and Type

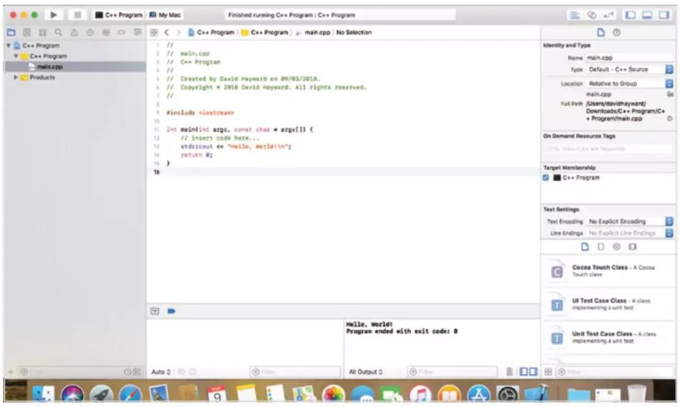
STEP 9 You can see that Xcode has automatically completed a basic Hello World program for you. While it may not make much sense at present, you will discover more as you progress, the content is just Xcode utilising what's available on the Mac.



```
//
//  main.cpp
//  C++ Program
//
//  Created by David Hayward on 09/03/2018.
//  Copyright © 2018 David Hayward. All rights reserved.
//

#include <iostream>

int main(int argc, const char * argv[]) {
    // insert code here...
    std::cout << "Hello, World!\n";
    return 0;
}
```

C++ Program: Ready | Today at 12:35
C++ Program
main.cpp
No Selection

STEP 10 When you want to run the code, click on Product > Run. You may be asked to enable Developer Mode on the Mac; this is to authorise Xcode to perform functions without needing your password every session. When the program executes, the output is displayed at the bottom of the Xcode window.



```
//
//  main.cpp
//  C++ Program
//
//  Created by David Hayward on 09/03/2018.
//  Copyright © 2018 David Hayward. All rights reserved.
//

#include <iostream>

int main(int argc, const char * argv[]) {
    // insert code here...
    std::cout << "Hello, World!\n";
    return 0;
}
```

```
Hello, World!
Program ended with exit code: 0
```

Finished running C++ Program : C++ Program
All Output

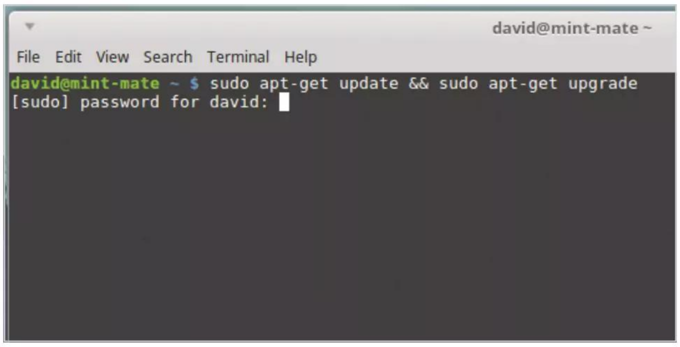
How to Set Up C++ in Linux

Linux is a great C++ coding environment. Most Linux distros already have the essential components preinstalled, such as a compiler and the text editors are excellent for entering code into, including colour coding; and there's tons of extra software available to help you out.

LINUX++

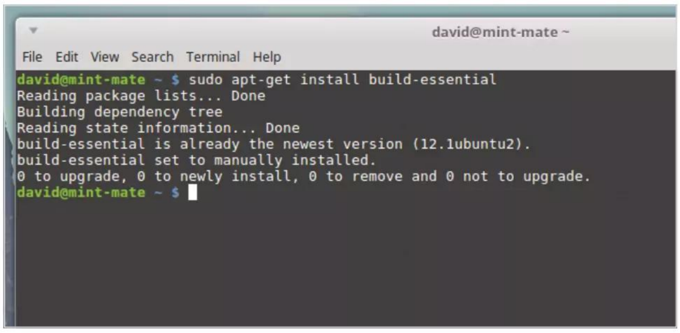
If you're not familiar with Linux, then we recommend taking a look at one of our Linux titles from the BDM Publications range. If you have a Raspberry Pi, the commands used below work just fine and for this example we're using Linux Mint.

STEP 1 The first step is to ensure Linux is ready for your C++ code, so check the system and software are up to date. Open a Terminal and enter: **sudo apt-get update && sudo apt-get upgrade**. Then press Return and enter your password. These commands update the entire system and any installed software.


david@mint-mate ~
File Edit View Search Terminal Help

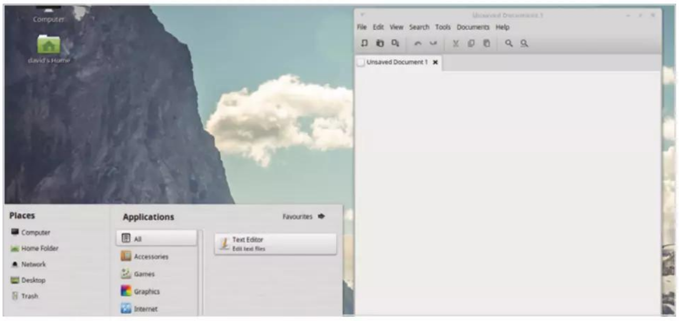
```
david@mint-mate ~ $ sudo apt-get update && sudo apt-get upgrade
[sudo] password for david: 
```

STEP 2 Most Linux distros come preinstalled with all the necessary components to start coding in C++; however, it's always worth checking to see if everything is present. Still within the Terminal, enter: s**udo apt-get install build-essential** and press Return. If you have the right components nothing is installed; if you're missing some then they are installed by the command.


david@mint-mate ~
File Edit View Search Terminal Help

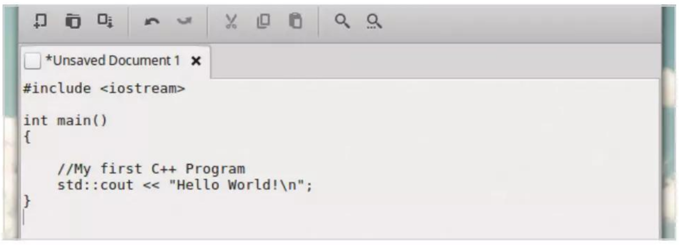
```
david@mint-mate ~ $ sudo apt-get install build-essential
Reading package lists... Done
Building dependency tree
Reading state information... Done
build-essential is already the newest version (12.1ubuntu2).
build-essential set to manually installed.
0 to upgrade, 0 to newly install, 0 to remove and 0 not to upgrade.
david@mint-mate ~ $ 
```

STEP 3 Amazingly, that's it. Everything is already for you to start coding. Here's how to get your first C++ program up and running. In Linux Mint the main text editor is Xed, which you can launch by clicking on the Menu and typing Xed into the search bar. Click on the Text Editor button in the right-hand pane to open it.


Computer
david's Home
File Edit View Search Tools Documents Help
Unsaved Document 1
Places
Computer
Home Folder
Network
Desktop
Trash
Applications
Favourites
All
Accessories
Games
Graphics
Internet
Text Editor
Edit text files

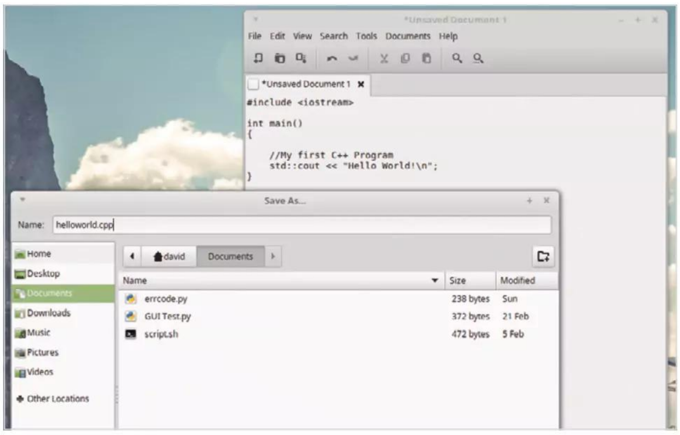
STEP 4 In Xed, or any other text editor you may be using, enter the lines of code that make up your C++ Hello World program. It's a little different to what the Mac produced:

```
#include <iostream>

int main()
{
//My first C++ program
std::cout <<  “Hello World!\n”;
}
```


*Unsaved Document 1

```
#include <iostream>

int main()
{

    //My first C++ Program
    std::cout << "Hello World!\n";
}
```

STEP 5 When you've entered your code, click File > Save As and choose a folder in which to save your program. Name the file as helloworld.cpp (it can be any name as long as it has .cpp as the extension). Click Save to continue.



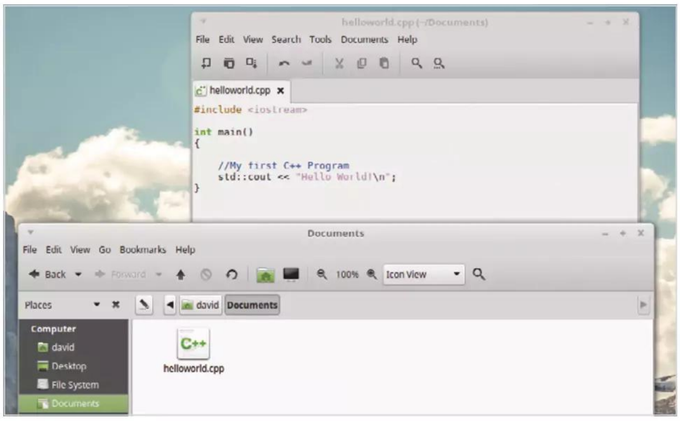
```
#include <iostream>

int main()
{

    //My first C++ Program
    std::cout << "Hello World!\n";
}
```

Name: helloworld.cpp

Name	Size	Modified
errcode.py	238 bytes	Sun
GUI Test.py	372 bytes	21 Feb
script.sh	472 bytes	5 Feb

*Unsaved Document 1
File Edit View Search Tools Documents Help
*Unsaved Document 1
Save As...
Home
Desktop
Documents
Downloads
Music
Pictures
Videos
Other Locations
david
Documents

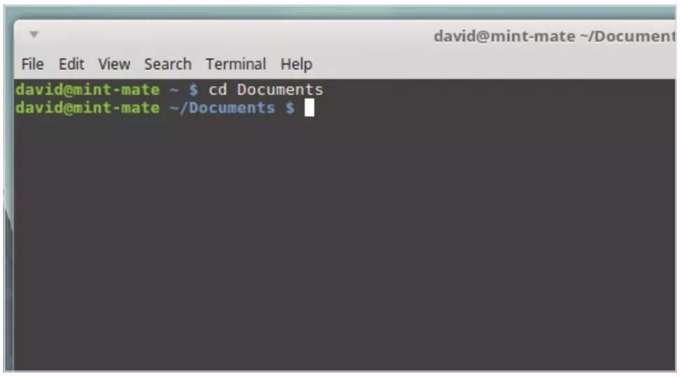
STEP 6 The first thing to notice is that Xed has automatically recognised this as a C++ file, since the file extension is now set to .cpp. The colour coding is present in the code and if you open up the file manager you can also see that file's icon has C++ stamped on it.



```
#include <iostream>

int main()
{

    //My first C++ Program
    std::cout << "Hello World!\n";
}
```

helloworld.cpp (~/Documents)
File Edit View Search Tools Documents Help
helloworld.cpp
Documents
File Edit View Go Bookmarks Help
Back
Forward
100%
Icon View
Places
david
Documents
Computer
david
Desktop
File System
Documents
C++
helloworld.cpp

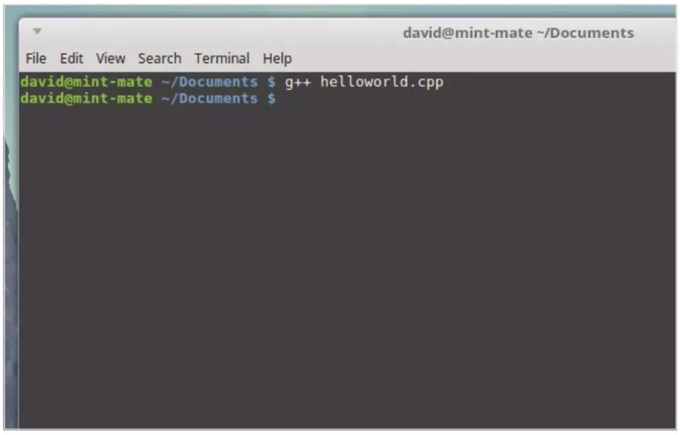
STEP 7 With your code now saved, drop into the Terminal again. You need to navigate to the location of the C++ file you've just saved. Our example is in the Documents folder, so we can navigate to it by entering: **cd Documents**. Remember, the Linux Terminal is case sensitive, so any capitals must be entered correctly.



```
david@mint-mate ~ $ cd Documents
david@mint-mate ~/Documents $ 
```

david@mint-mate ~/Document
File Edit View Search Terminal Help

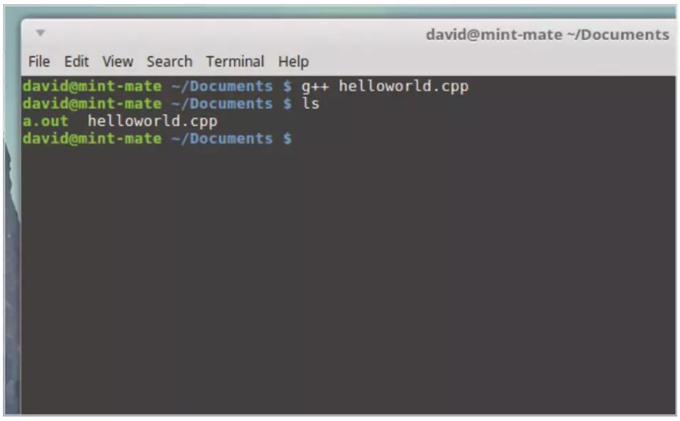
STEP 8 Before you can execute the C++ file you need to compile it. In Linux it's common to use g++, an open source C++ compiler; as you're now in the same folder as the C++ file, enter: **g++ helloworld.cpp** in the Terminal and press Return.



```
david@mint-mate ~/Documents $ g++ helloworld.cpp
david@mint-mate ~/Documents $
```

david@mint-mate ~/Documents
File Edit View Search Terminal Help

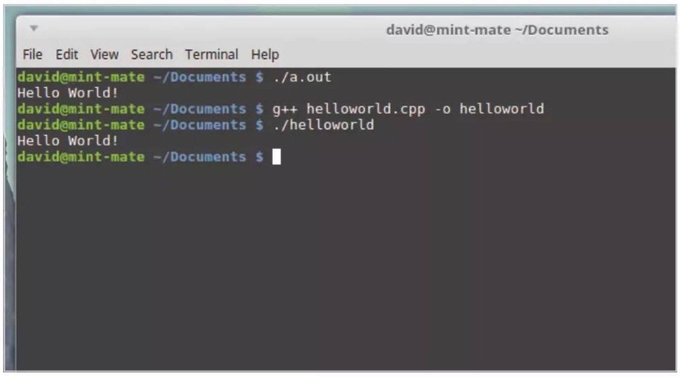
STEP 9 It takes a short time while the code is compiled by g++ but providing there are no mistakes or errors in the code you are returned to the command prompt. The compiling of the code has created a new file. If you enter: ls into the Terminal you can see that alongside your C++ file is a.out.



```
david@mint-mate ~/Documents $ g++ helloworld.cpp
david@mint-mate ~/Documents $ ls
a.out  helloworld.cpp
david@mint-mate ~/Documents $
```

david@mint-mate ~/Documents
File Edit View Search Terminal Help

STEP 10 The a.out file is the compiled C++ code. To run the code enter: **./a.out** and press Return. The words 'Hello World!' appear on the screen. However, a.out isn't very friendly. To name it something else post-compiling, you can recompile with: **g++ helloworld.cpp -o helloworld**. This creates an output file called helloworld which can be run with: **./helloworld**.



```
david@mint-mate ~/Documents $ ./a.out
Hello World!
david@mint-mate ~/Documents $ g++ helloworld.cpp -o helloworld
david@mint-mate ~/Documents $ ./helloworld
Hello World!
david@mint-mate ~/Documents $ 
```

david@mint-mate ~/Documents
File Edit View Search Terminal Help

Other C++ IDEs to Install

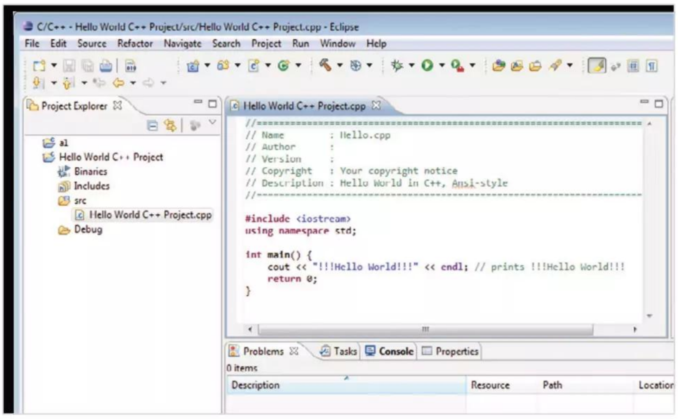
If you want to try a different approach to working with your C++ code, then there are plenty of options available to you. Windows is the most prolific platform for C++ IDEs but there are plenty for Mac and Linux users too.

DEVELOPING C++

Here are ten great C++ IDEs that are worth looking into. You can install one or all of them if you like, but find the one that works best for you.

ECLIPSE

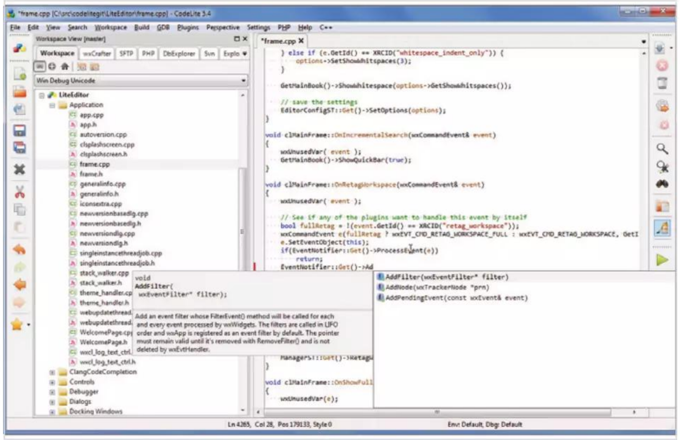
Eclipse is a hugely popular C++ IDE that offers the programmer a wealth of features. It has a great, clean interface, is easy to use and available for Windows, Linux and Mac. Head over to **www.eclipse.org/downloads/** to download the latest version. If you're stuck, click the Need Help link for more information.



```
//============================================================================
// Name        : Hello.cpp
// Author      :
// Version     :
// Copyright   : Your copyright notice
// Description : Hello World in C++, Ansi-style
//============================================================================

#include <iostream>
using namespace std;

int main() {
	cout << "!!!Hello World!!!" << endl; // prints !!!Hello World!!!
	return 0;
}
```

CODELITE

CodeLite is a free and open source IDE that's regularly updated and available for Windows, Linux and macOS. It's lightweight, uncomplicated and extremely powerful. You can find out more information as well as how to download and install it at **www.codelite.org/**.



```
    } else if (e.GetId() == XRCID("whitespace_indent_only")) {
        options->SetShowWhitspaces(3);
    }

    GetMainBook()->ShowWhitespace(options->GetShowWhitspaces());

    // save the settings
    EditorConfigST::Get()->SetOptions(options);
}

void clMainFrame::OnIncrementalSearch(wxCommandEvent& event)
{
    wxUnusedVar( event );
    GetMainBook()->ShowQuickBar(true);
}

void clMainFrame::OnRetagWorkspace(wxCommandEvent& event)
{
    wxUnusedVar( event );

    // See if any of the plugins want to handle this event by itself
    bool fullRetag = !(event.GetId() == XRCID("retag_workspace"));
    wxCommandEvent e(fullRetag ? wxEVT_CMD_RETAG_WORKSPACE_FULL : wxEVT_CMD_RETAG_WORKSPACE, GetI
    e.SetEventObject(this);
    if(EventNotifier::Get()->ProcessEvent(e))
        return;
    EventNotifier::Get()->Ad
```

```
AddFilter(wxEventFilter* filter)
AddNode(wxTrackerNode *prn)
AddPendingEvent(const wxEvent& event)
```

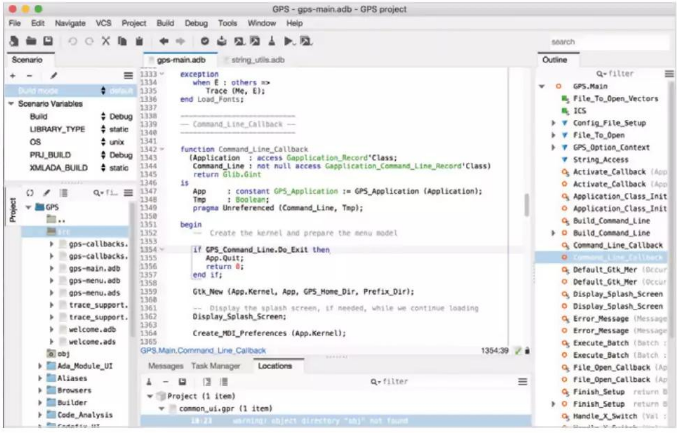
Add an event filter whose FilterEvent() method will be called for each and every event processed by wxWidgets. The filters are called in LIFO order and wxApp is registered as an event filter by default. The pointer must remain valid until it's removed with RemoveFilter() and is not deleted by wxEvtHandler.

Ln 4265, Col 28, Pos 179133, Style 0

Env: Default, Dbg: Default

GNAT

The GNAT Programming Studio (GPS) is a powerful and intuitive IDE that supports testing, debugging and code analysis. The Community Edition is free, whereas the Pro version costs; however, the Community Edition is available for Windows, Mac, Linux and even the Raspberry Pi. You can find it at **www.adacore.com/download**.



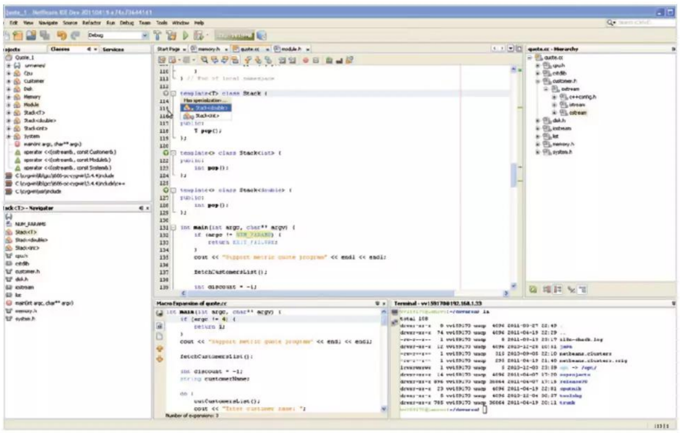
```
      exception
         when E : others =>
            Trace (Me, E);
      end Load_Fonts;

      ---------------------------
      -- Command_Line_Callback --
      ---------------------------

      function Command_Line_Callback
        (Application  : access Gapplication_Record'Class;
         Command_Line : not null access Gapplication_Command_Line_Record'Class)
         return Glib.Gint
      is
         App     : constant GPS_Application := GPS_Application (Application);
         Tmp     : Boolean;
         pragma Unreferenced (Command_Line, Tmp);

      begin
         --  Create the kernel and prepare the menu model

         if GPS_Command_Line.Do_Exit then
            App.Quit;
            return 0;
         end if;

         Gtk_New (App.Kernel, App, GPS_Home_Dir, Prefix_Dir);

         --  Display the splash screen, if needed, while we continue loading
         Display_Splash_Screen;

         Create_MDI_Preferences (App.Kernel);
```

GPS.Main.Command_Line_Callback 1354:39

```
18:21    warning: object directory "obj" not found
```

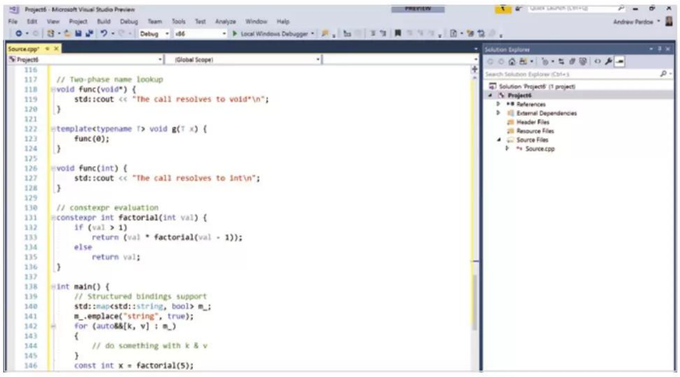
NETBEANS

Another popular choice is NetBeans. This is another excellent IDE that's packed with features and a pleasure to use. NetBeans IDE includes project based templates for C++ that give you the ability to build applications with dynamic and static libraries. Find out more at **www.netbeans.org/features/cpp/index.html.**



VISUAL STUDIO

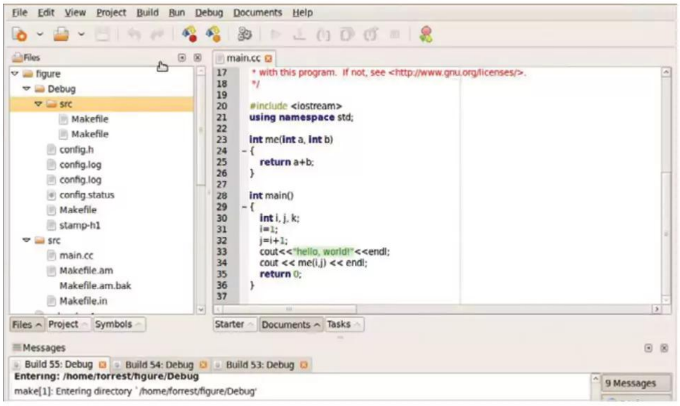
Microsoft's Visual Studio is a mammoth C++ IDE that allows you to create applications for Windows, Android, iOS and the web. The Community version is free to download and install but the other versions allow a free trial period. Go to **www.visualstudio.com/** to see what it can do for you.


Project6 - Microsoft Visual Studio Preview
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug
x86
Local Windows Debugger
Source.cpp*
Project6
(Global Scope)
Solution Explorer

```
// Two-phase name lookup
void func(void*) {
    std::cout << "The call resolves to void*\n";
}

template<typename T> void g(T x) {
    func(0);
}

void func(int) {
    std::cout << "The call resolves to int\n";
}

// constexpr evaluation
constexpr int factorial(int val) {
    if (val > 1)
        return (val * factorial(val - 1));
    else
        return val;
}

int main() {
    // Structured bindings support
    std::map<std::string, bool> m_;
    m_.emplace("string", true);
    for (auto&&[k, v] : m_)
    {
        // do something with k & v
    }
    const int x = factorial(5);
```

ANJUTA

The Anjuta DevStudio is a Linux-only IDE that features some of the more advanced features you would normally find in a paid software development studio. There's a GUI designer, source editor, app wizard, interactive debugger and much more. Go to **www.anjuta.org/** for more information.

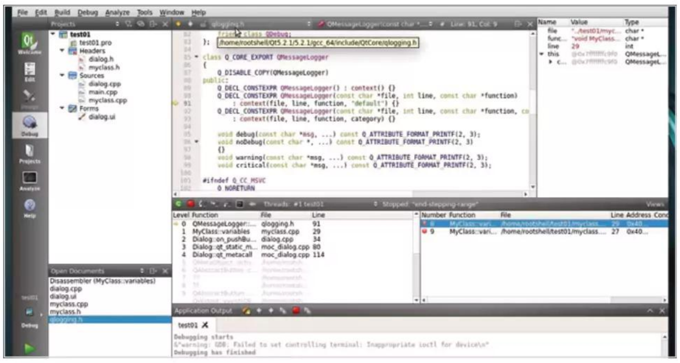

File Edit View Project Build Run Debug Documents Help
Files
figure
Debug
src
Makefile
Makefile
config.h
config.log
config.log
config.status
Makefile
stamp-h1
src
main.cc
Makefile.am
Makefile.am.bak
Makefile.in
Files
Project
Symbols
main.cc
Starter
Documents
Tasks
Messages
Build 55: Debug
Build 54: Debug
Build 53: Debug
9 Messages

```
 * with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

#include <iostream>
using namespace std;

int me(int a, int b)
{
    return a+b;
}

int main()
{
    int i, j, k;
    i=1;
    j=i+1;
    cout<<"hello, world!"<<endl;
    cout << me(i,j) << endl;
    return 0;
}
```

```
Entering: /home/forrest/figure/Debug
make[1]: Entering directory `/home/forrest/figure/Debug'
```

QT CREATOR

This cross-platform IDE is designed to create C++ applications for desktop and mobile environments. It comes with a code editor and integrated tools for testing and debugging, as well as deploying to you chosen platform. It's not free but there is a trial period on offer before requiring purchasing: **www.qt.io/qt-features-libraries-apis-tools-and-ide/**.


File Edit Build Debug Analyze Tools Window Help
Projects
test01
test01.pro
Headers
dialog.h
myclass.h
Sources
dialog.cpp
main.cpp
myclass.cpp
Forms
dialog.ui
Welcome
Edit
Design
Debug
Projects
Analyze
Help
Open Documents
dialog.cpp
dialog.ui
myclass.cpp
myclass.h
qlogging.h
qlogging.h
QMessageLogger(const char *...
Line: 91, Col: 9
/home/rootshell/Qt5.2.1/5.2.1/gcc_64/include/QtCore/qlogging.h
Name
Value
Type
Threads: #1 test01
Stopped: "end-stepping-range"
Application Output
test01

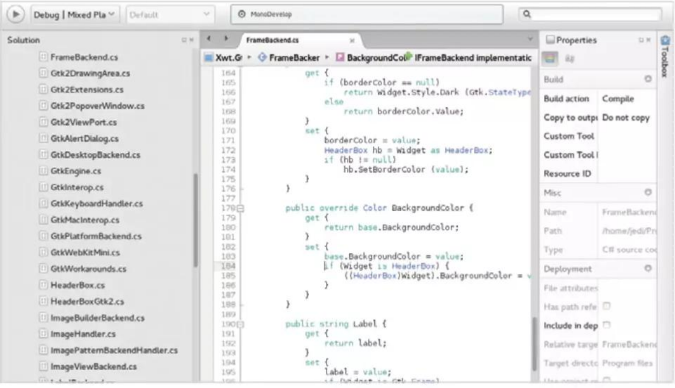
```
    friend class QDebug;
};

class Q_CORE_EXPORT QMessageLogger
{
    Q_DISABLE_COPY(QMessageLogger)
public:
    Q_DECL_CONSTEXPR QMessageLogger() : context() {}
    Q_DECL_CONSTEXPR QMessageLogger(const char *file, int line, const char *function)
        : context(file, line, function, "default") {}
    Q_DECL_CONSTEXPR QMessageLogger(const char *file, int line, const char *function, co
        : context(file, line, function, category) {}

    void debug(const char *msg, ...) const Q_ATTRIBUTE_FORMAT_PRINTF(2, 3);
    void noDebug(const char *, ...) const Q_ATTRIBUTE_FORMAT_PRINTF(2, 3)
    {}
    void warning(const char *msg, ...) const Q_ATTRIBUTE_FORMAT_PRINTF(2, 3);
    void critical(const char *msg, ...) const Q_ATTRIBUTE_FORMAT_PRINTF(2, 3);

#ifndef Q_CC_MSVC
    Q_NORETURN
```

Level	Function	File	Line
0	QMessageLogger::...	qlogging.h	91
1	MyClass::variables	myclass.cpp	29
2	Dialog::on_pushBu...	dialog.cpp	34
3	Dialog::qt_static_m...	moc_dialog.cpp	80
4	Dialog::qt_metacall	moc_dialog.cpp	114

```
Debugging starts
&"warning: GDB: Failed to set controlling terminal: Inappropriate ioctl for device\n"
Debugging has finished
```

MONODEVELOP

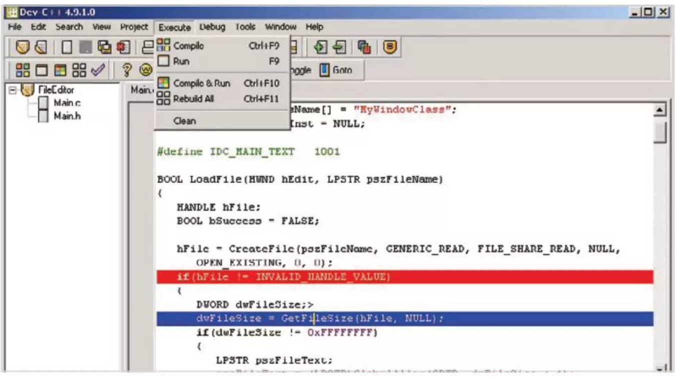
This excellent IDE allows developers to write C++ code for desktop and web applications across all the major platforms. There's an advanced text editor, integrated debugger and a configurable workbench to help you create your code. It's available for Windows, Mac and Linux and is free to download and use: **www.monodevelop.com/**.


Debug | Mixed Pla
Default
MonoDevelop
Solution
FrameBackend.cs
Gtk2DrawingArea.cs
Gtk2Extensions.cs
Gtk2PopoverWindow.cs
Gtk2ViewPort.cs
GtkAlertDialog.cs
GtkDesktopBackend.cs
GtkEngine.cs
GtkInterop.cs
GtkKeyboardHandler.cs
GtkMacInterop.cs
GtkPlatformBackend.cs
GtkWebKitMini.cs
GtkWorkarounds.cs
HeaderBox.cs
HeaderBoxGtk2.cs
ImageBuilderBackend.cs
ImageHandler.cs
ImagePatternBackendHandler.cs
ImageViewBackend.cs
FrameBackend.cs
Xwt.G
FrameBacker
BackgroundCol
IFrameBackend implementatic
Properties
Toolbox
Build
Build action
Compile
Copy to outpu
Do not copy
Custom Tool
Custom Tool
Resource ID
Misc
Name
FrameBackend
Path
/home/jedi/Pr
Type
C# source co
Deployment
File attributes
Has path refe
Include in dep
Relative targe
FrameBackend
Target directo
Program files

```
            get {
                if (borderColor == null)
                    return Widget.Style.Dark (Gtk.StateType
                else
                    return borderColor.Value;
            }
            set {
                borderColor = value;
                HeaderBox hb = Widget as HeaderBox;
                if (hb != null)
                    hb.SetBorderColor (value);
            }
        }

        public override Color BackgroundColor {
            get {
                return base.BackgroundColor;
            }
            set {
                base.BackgroundColor = value;
                if (Widget is HeaderBox) {
                    ((HeaderBox)Widget).BackgroundColor = v
                }
            }
        }

        public string Label {
            get {
                return label;
            }
            set {
                label = value;
```

DEV C++

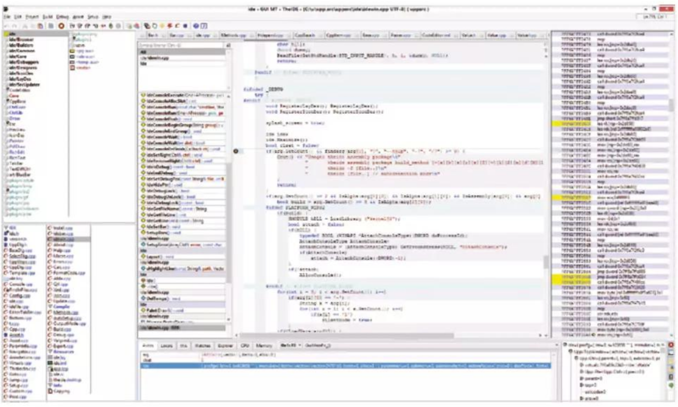
Bloodshed Dev C++, despite its colourful name, is an older IDE that is for Windows systems only. However, many users praise its clean interface and uncomplicated way of coding and compiling. Although there's not been much updating for some time, it's certainly one to consider if you want something different: **www.bloodshed.net/devcpp.html**.


Dev C++ 4.9.1.0
File Edit Search View Project Execute Debug Tools Window Help
Compile Ctrl+F9
Run F9
Compile & Run Ctrl+F10
Rebuild All Ctrl+F11
Clean
oggle
Goto
FileEditor
Main.c
Main.h
Main.

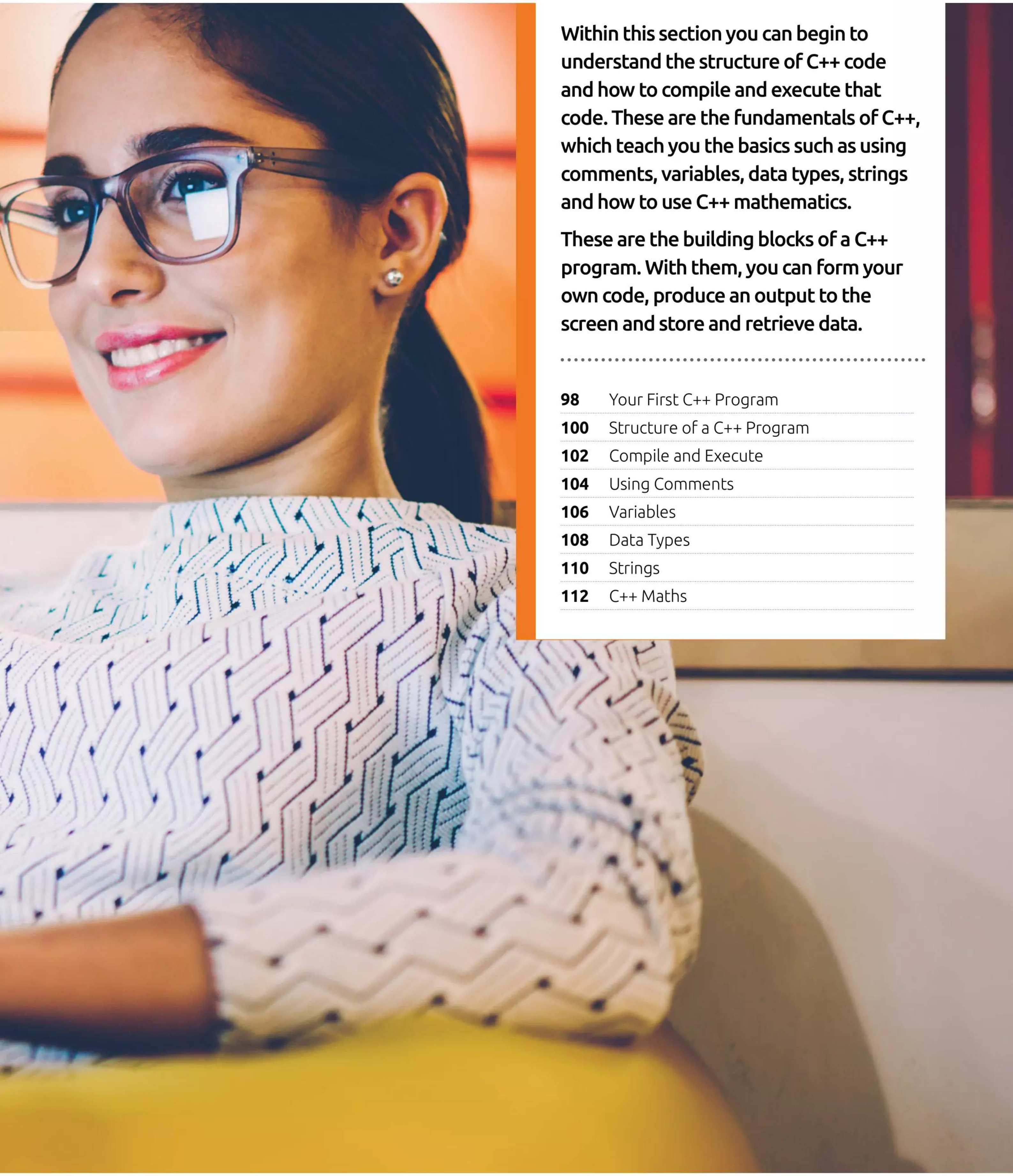
```
eName[] = "MyWindowClass";
Inst = NULL;

#define IDC_MAIN_TEXT   1001

BOOL LoadFile(HWND hEdit, LPSTR pszFileName)
{
   HANDLE hFile;
   BOOL bSuccess = FALSE;

   hFile = CreateFile(pszFileName, GENERIC_READ, FILE_SHARE_READ, NULL,
      OPEN_EXISTING, 0, 0);
   if(hFile != INVALID_HANDLE_VALUE)
   {
      DWORD dwFileSize;>
      dwFileSize = GetFileSize(hFile, NULL);
      if(dwFileSize != 0xFFFFFFFF)
      {
         LPSTR pszFileText;
```

U++

Ultimate++ is a cross-platform C++ IDE that boats a rapid development of code through the smart and aggressive use of C++. For the novice, it's a beast of an IDE but behind its complexity is a beauty that would make a developer's knees go wobbly. Find out more at **www.ultimatepp.org/index.html**.



C++ Fundamentals



Within this section you can begin to understand the structure of C++ code and how to compile and execute that code. These are the fundamentals of C++, which teach you the basics such as using comments, variables, data types, strings and how to use C++ mathematics.

These are the building blocks of a C++ program. With them, you can form your own code, produce an output to the screen and store and retrieve data.

98 Your First C++ Program

100 Structure of a C++ Program

102 Compile and Execute

104 Using Comments

106 Variables

108 Data Types

110 Strings

112 C++ Maths

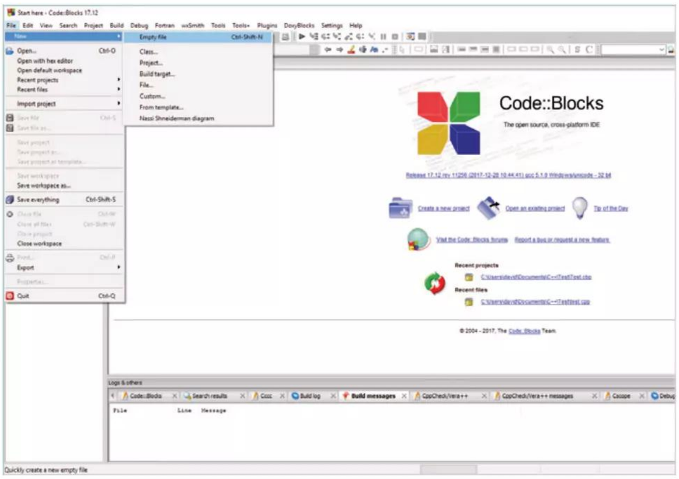
Your First C++ Program

You may have followed the Mac and Linux examples previously but you're going to be working exclusively in Windows and Code::Blocks from here on. Let's begin by writing your first C++ program and taking the first small step into a larger coding world.

HELLO, WORLD!

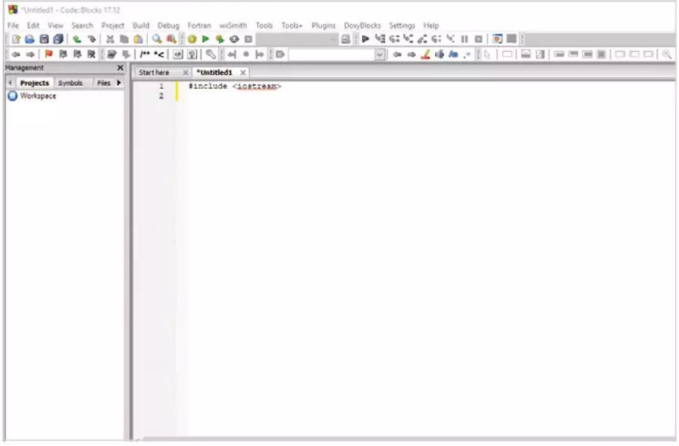
It's traditional in programming for the first code to be entered to output the words 'Hello, World!' to the screen. Interestingly, this dates back to 1968 using a language called BCPL.

STEP 1 As mentioned, we're using Windows 10 and the latest version of Code::Blocks for the rest of the C++ code in this book. Begin by launching Code::Blocks. When open, click on File > New > Empty File or press Ctrl+Shift+N on the keyboard.


Start here - Code::Blocks 17.12
New
Empty file
Ctrl-Shift-N
Class...
Project...
Build target...
File...
Custom...
From template...
Nassi Shneiderman diagram
Code::Blocks
The open source, cross-platform IDE
Quickly create a new empty file

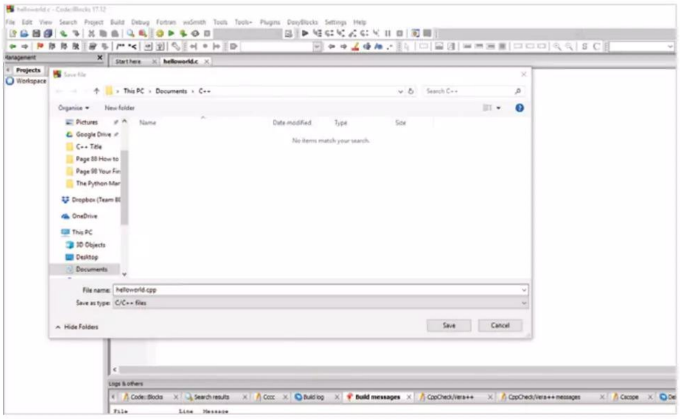
STEP 2 Now you can see a blank screen, with the tab labelled *Untitled1, and the number one in the top left of the main Code::Blocks window. Begin by clicking in the main window, so the cursor is next to the number one, and entering:

```
#include <iostream>
```


*Untitled1 - Code::Blocks 17.12
Start here
*Untitled1

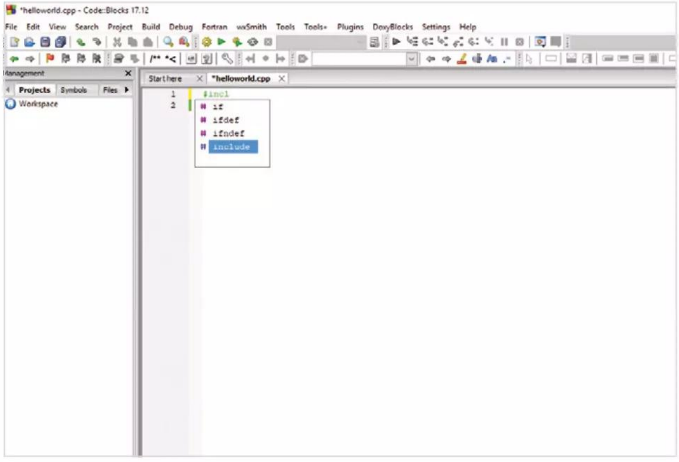
```
#include <iostream>
```

STEP 3 At the moment it doesn't look like much, and it makes even less sense, but we'll get to that in due course. Now click on File > Save File As. Create or find a suitable location on your hard drive and in the File Name box, call it helloworld.cpp. Click the Save as type box and select C/C++ files. Click the Save button.


Save file
This PC › Documents › C++
No items match your search.
Save
Cancel

File name: helloworld.cpp
Save as type: C/C++ files

STEP 4 You can see that Code::Blocks has now changed the colour coding, recognising that the file is now C++ code. This means that code can be auto-selected from the Code::Blocks repository. Delete the #include <iostream> line and re-enter it. You can see the auto-select boxes appearing.


*helloworld.cpp - Code::Blocks 17.12
Start here
*helloworld.cpp
if
ifdef
ifndef
include

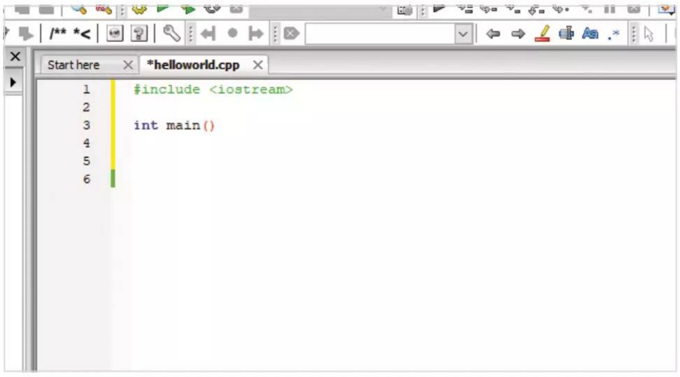
```
#incl
```

STEP 5

Auto-selection of commands is extremely handy and cuts out potential mistyping. Press Return to get to line 3, then enter:

`int main()`

Note: there's no space between the brackets.


Start here
*helloworld.cpp

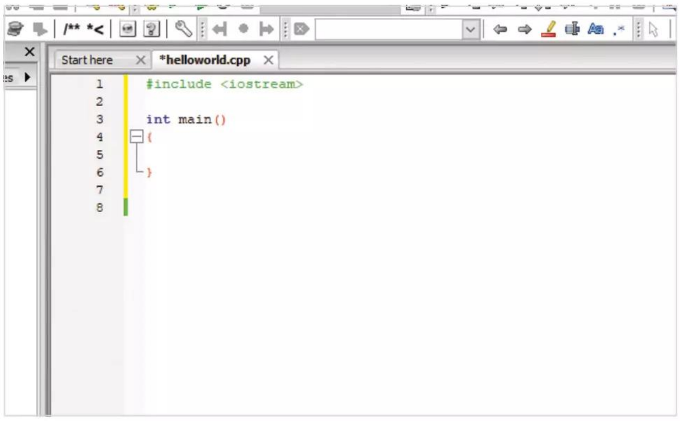
```
1   #include <iostream>
2
3   int main()
4
5
6
```

STEP 6

On the next line below int main(), enter a curly bracket:

`{`

This can be done by pressing Shift and the key to the right of P on an English UK keyboard layout.

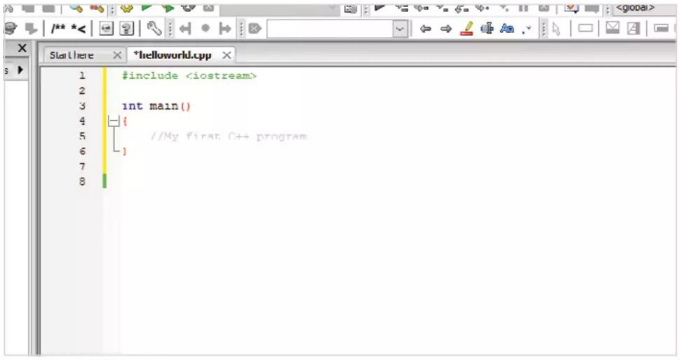

Start here
*helloworld.cpp

```
1   #include <iostream>
2
3   int main()
4   {
5
6   }
7
8
```

STEP 7

Notice that Code::Blocks has automatically created a corresponding closing curly bracket a couple of lines below, linking the pair, as well as a slight indent. This is due to the structure of C++ and it's where the meat of the code is entered. Now enter:

`//My first C++ program`

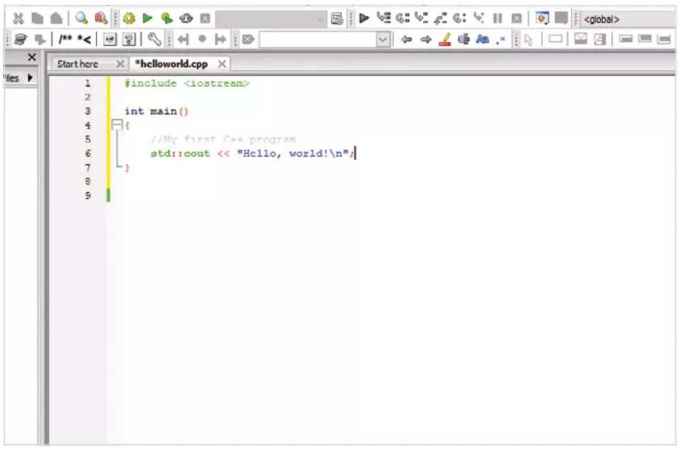

Start here
*helloworld.cpp
<global>

```
1   #include <iostream>
2
3   int main()
4   {
5       //My first C++ program
6   }
7
8
```

STEP 8

Note again the colour coding change. Press Return at the end of the previous step's line, and then enter:

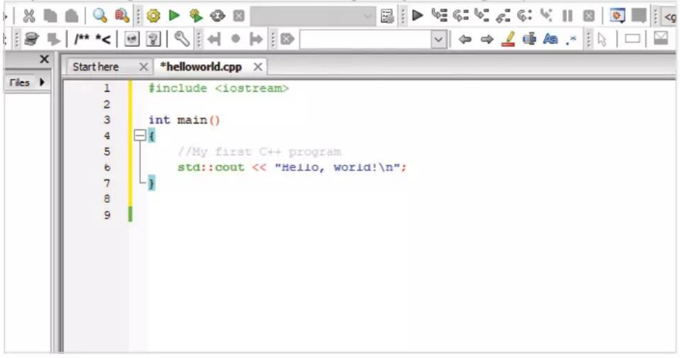
`std::cout << “Hello, world!\n”;`


Start here
*helloworld.cpp
<global>

```
1   #include <iostream>
2
3   int main()
4   {
5       //My first C++ program
6       std::cout << "Hello, world!\n";
7   }
8
9
```

STEP 9

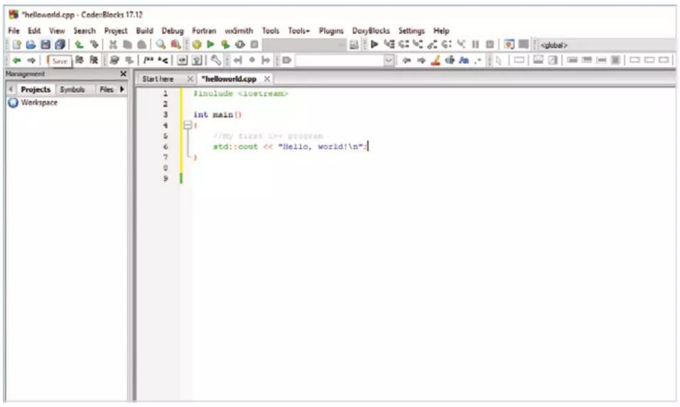
Just as before, Code::Blocks auto-completes the code you're entering, including placing a closing speech mark as soon as you enter the first. Don't forget the semicolon at the end of the line; this is one of the most important elements to a C++ program and we'll tell you why in the next section. For now, move the cursor down to the closing curly bracket and press Return.


Files
Start here
*helloworld.cpp

```
1   #include <iostream>
2
3   int main()
4   {
5       //My first C++ program
6       std::cout << "Hello, world!\n";
7   }
8
9
```

STEP 10

That's all you need to do for the moment. It may not look terribly amazing but C++ is best absorbed in small chunks. Don't execute the code at the moment as you need to look at how a C++ program is structured first; then you can build and run the code. For now, click on Save, the single floppy disc icon.


*helloworld.cpp - Code::Blocks 17.12
File
Edit
View
Search
Project
Build
Debug
Fortran
wxSmith
Tools
Tools+
Plugins
DoxyBlocks
Settings
Help
Save
<global>
Management
Projects
Symbols
Files
Workspace
Start here
*helloworld.cpp

```
1   #include <iostream>
2
3   int main()
4   {
5       //My first C++ program
6       std::cout << "Hello, world!\n";
7   }
8
9
```

Structure of a C++ Program

C++ has a very defined structure and way of doing things. Miss something out, even as small as a semicolon, and your entire program will fail to be compiled and executed. Many a professional programmer has fallen foul of sloppy structure.

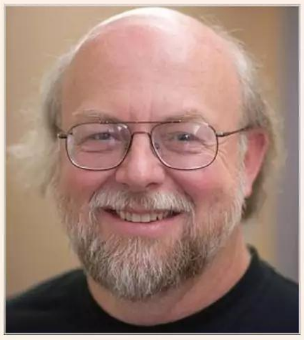
#INCLUDE <C++ STRUCTURE>

Learning the basics of programming, you begin to understand the structure of a program. The commands may be different from one language to the next, but you will start to see how the code works.

C++

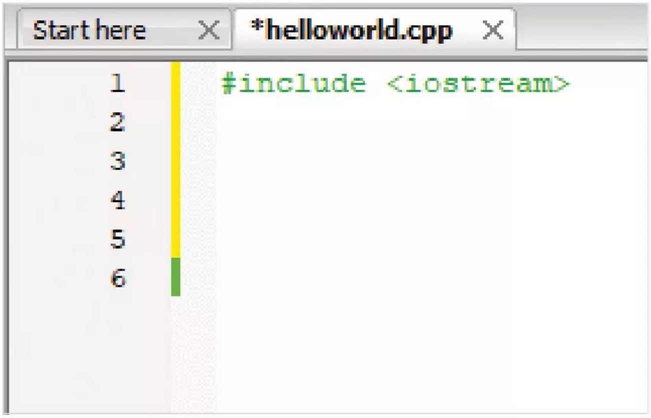
C++ was invented by Danish student Bjarne Stroustrup in 1979, as a part of his Ph.D. thesis. Initially C++ was called C with Classes, which added features to the already popular C programming language, while making it a more user-friendly environment through a new structure.

Bjarne Stroustrup, inventor of C++.



#INCLUDE

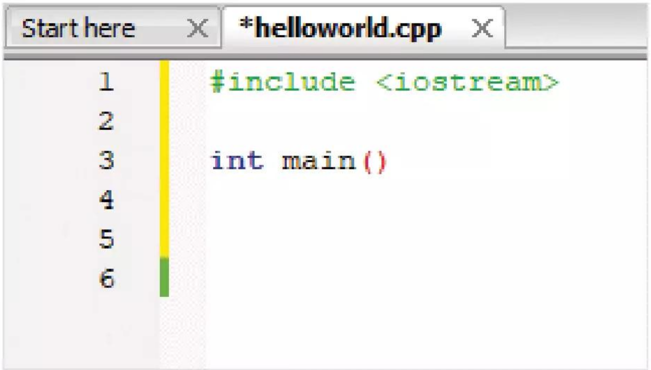
The structure of a C++ program is quite precise. Every C++ code begins with a directive: **#include <>**. The directive instructs the pre-processor to include a section of the standard C++ code. For example: **#include <iostream>** includes the iostream header to support input/output operations.



```
#include <iostream>
```

INT MAIN()

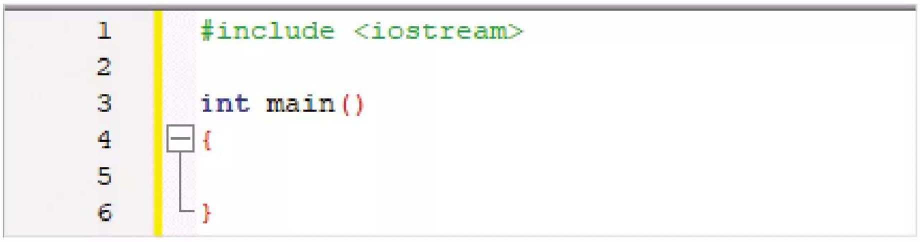
int main() initiates the declaration of a function, which is a group of code statements under the name 'main'. All C++ code begins at the main function, regardless of where it actually lies within the code.



```
#include <iostream>

int main()
```

BRACES

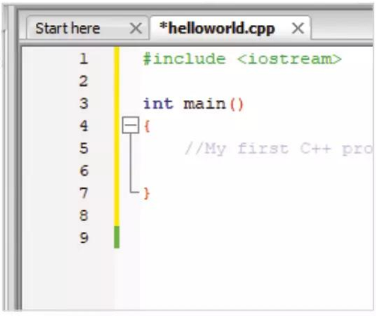
The open brace (curly brackets) is something that you may not have come across before, especially if you're used to Python. The open brace indicates the beginning of the main function and contains all the code that belongs to that function.



```
#include <iostream>

int main()
{

}
```

COMMENTS

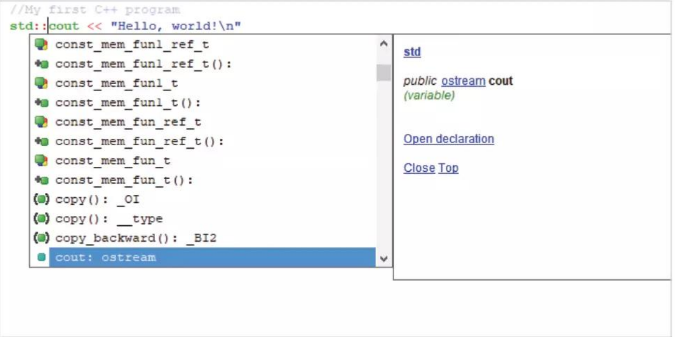
Lines that begin with a double slash are comments. This means they won't be executed in the code and are ignored by the compiler. Comments are designed to help you, or another programmer looking at your code, explain what's going on. There are two types of comment: /* covers multiple line comments, // a single line. Lines that begin with a double slash are comments. This means they won't be executed in the code and are ignored by the compiler. Comments are designed to help you, or another programmer looking at your code, explain what's going on. There are two types of comment: /* covers multiple line comments, // a single line.


Start here
*helloworld.cpp

```
#include <iostream>

int main()
{
    //My first C++ pro
}
```

<<

The two chevrons used here are insertion operators. This means that whatever follows the chevrons is to be inserted into the std::cout statement. In this case they're the words 'Hello, world', which are to be displayed on the screen when you compile and execute the code.



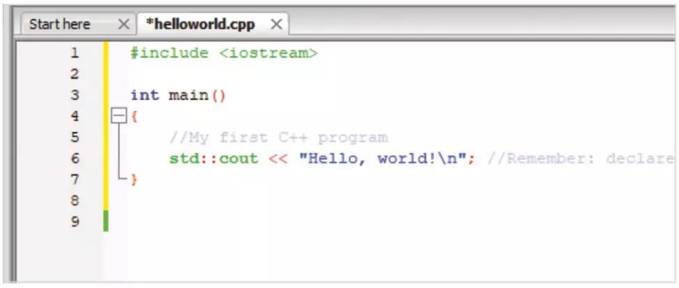
```
//My first C++ program
std::cout << "Hello, world!\n"
```

const_mem_funl_ref_t
const_mem_funl_ref_t():
const_mem_funl_t
const_mem_funl_t():
const_mem_fun_ref_t
const_mem_fun_ref_t():
const_mem_fun_t
const_mem_fun_t():
copy(): _OI
copy(): __type
copy_backward(): _BI2
cout: ostream
std
public ostream cout
(variable)
Open declaration
Close Top

STD

While **std** stands for something quite different, in C++ it means Standard. It's part of the Standard Namespace in C++, which covers a number of different statements and commands. You can leave the **std::** part out of the code but it must be declared at the start with: **using namespace std**; not both. For example:

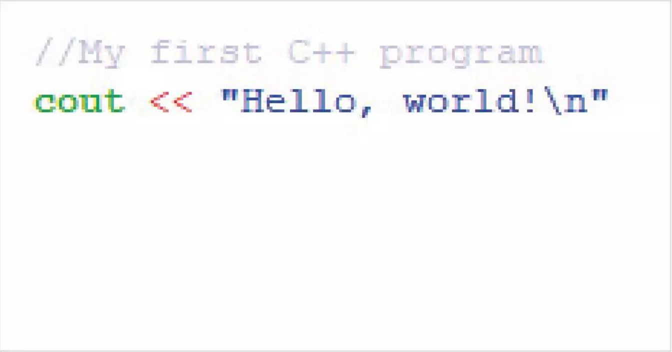
```
#include <iostream>
using namespace std;
```


Start here
*helloworld.cpp

```
#include <iostream>

int main()
{
    //My first C++ program
    std::cout << "Hello, world!\n"; //Remember: declare
}
```

OUTPUTS

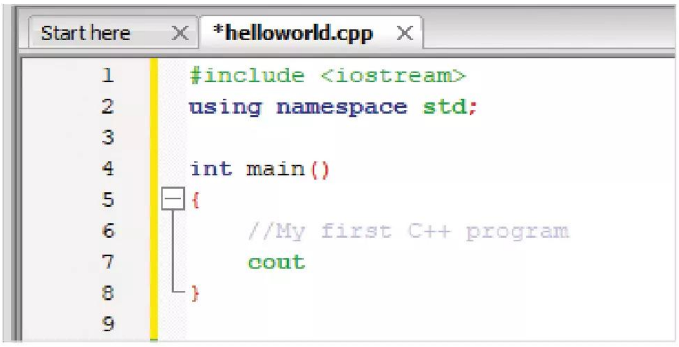
Leading on, the "Hello, world!" part is what we want to appear on the screen when the code is executed. You can enter whatever you like, as long as it's inside the quotation marks. The brackets aren't needed but some compilers insist on them. The \n part indicates a new line is to be inserted.



```
//My first C++ program
cout << "Hello, world!\n"
```

COUT

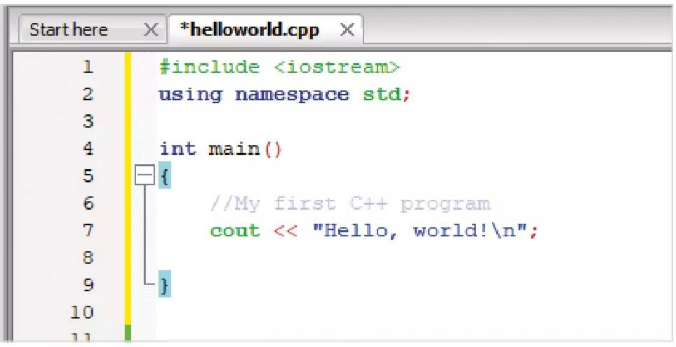
In this example we're using cout, which is a part of the Standard Namespace, hence why it's there, as you're asking C++ to use it from that particular namespace. Cout means Character OUTput, which displays, or prints, something to the screen. If we leave **std::** out we have to declare it at the start of the code, as mentioned previously.


Start here
*helloworld.cpp

```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout
}
```

; AND }

Finally you can see that lines within a function code block (except comments) end with a semicolon. This marks the end of the statement and all statements in C++ must have one at the end or the compiler fails to build the code. The very last line has the closing brace to indicate the end of the main function.


Start here
*helloworld.cpp

```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout << "Hello, world!\n";

}
```

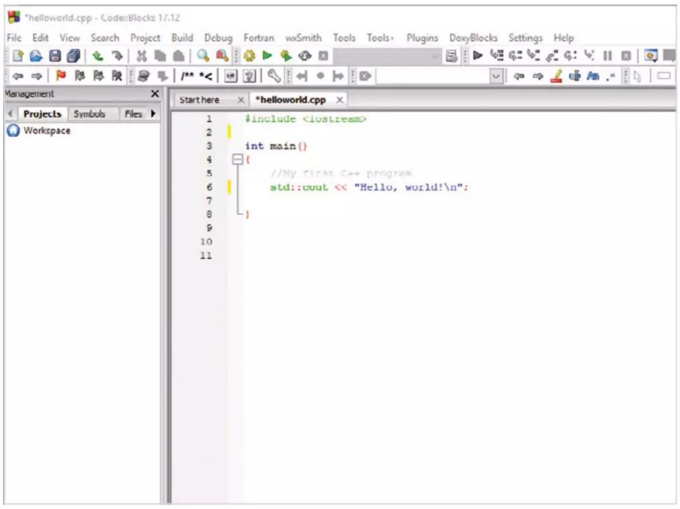
Compile and Execute

You've created your first C++ program and you now understand the basics behind the structure of one. Let's actually get things moving and compile and execute, or run if you prefer, the program and see how it looks.

GREETINGS FROM C++

Compiling and executing C++ code from Code::Blocks is extraordinarily easy; just a matter of clicking an icon and seeing the result. Here's how it's done.

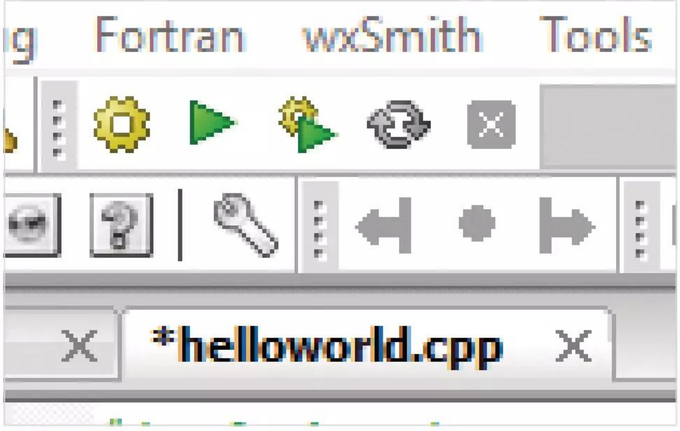
STEP 1 Open Code::Blocks, if you haven't already, and load up the previously saved Hello World code you created. Ensure that there are no visible errors, such as missing semicolons at the end of the std::cout line.



```
#include <iostream>

int main()
{
    //My first C++ program
    std::cout << "Hello, world!\n";

}
```

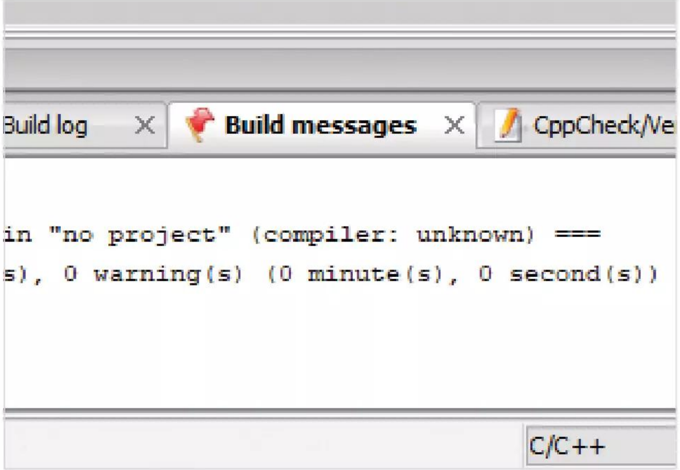
*helloworld.cpp - Code::Blocks 17.12
File
Edit
View
Search
Project
Build
Debug
Fortran
wxSmith
Tools
Tools+
Plugins
DoxyBlocks
Settings
Help
Management
Projects
Symbols
Files
Workspace
Start here
*helloworld.cpp

STEP 2 If your code is looking similar to the one in our screenshot, then look to the menu bar along the top of the screen. Under the Fortran entry in the topmost menu you can see a group of icons: a yellow cog, green play button and a cog/play button together. These are Build, Run, Build and Run functions.



g
Fortran
wxSmith
Tools
*helloworld.cpp

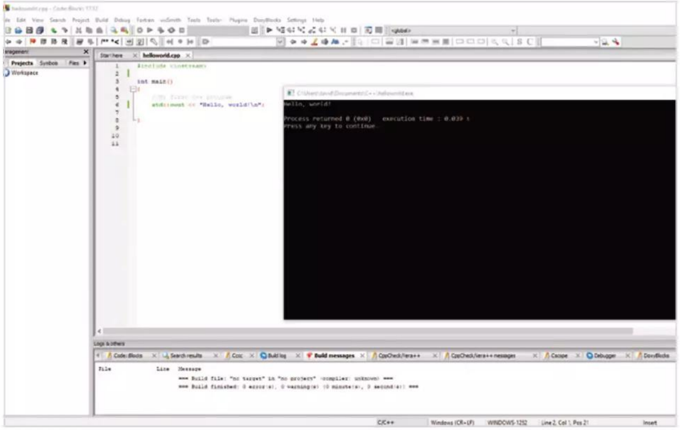
STEP 3 Start by clicking on the Build icon, the yellow cog. At this point, your code has now been run through the Code::Blocks compiler and checked for any errors. You can see the results of the Build by looking to the bottom window pane. Any messages regarding the quality of the code are displayed here.



```
in "no project" (compiler: unknown) ===
s), 0 warning(s) (0 minute(s), 0 second(s))
```

Build log
Build messages
CppCheck/Ve
C/C++

STEP 4 Now click on the Run icon, the green play button. A command line box appears on your screen displaying the words: Hello, world!, followed by the time it's taken to execute the code, and asking you press a key to continue. Well done, you just compiled and executed your first C++ program.



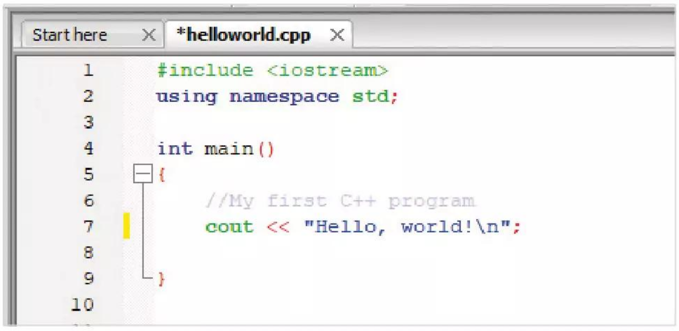
STEP 5

Pressing any key in the command line box closes it, returning you to Code::Blocks. Let's alter the code slightly. Under the #include line, enter:

```
using namespace std;
```

Then, delete the std:: part of the Cout line; like so:

```
cout << “Hello, world\n”;
```

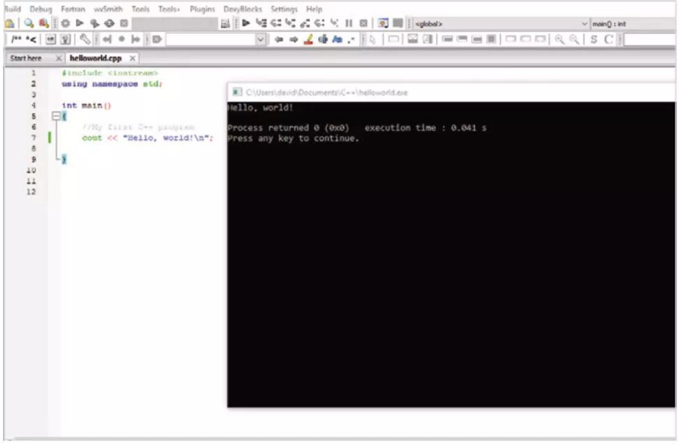


Start here
*helloworld.cpp

```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout << "Hello, world!\n";
}
```

STEP 6

In order to apply the new changes to the code, you need to re-compile, build, and run it again. This time, however, you can simply click the Build/Run icon, the combined yellow cog and green play button.



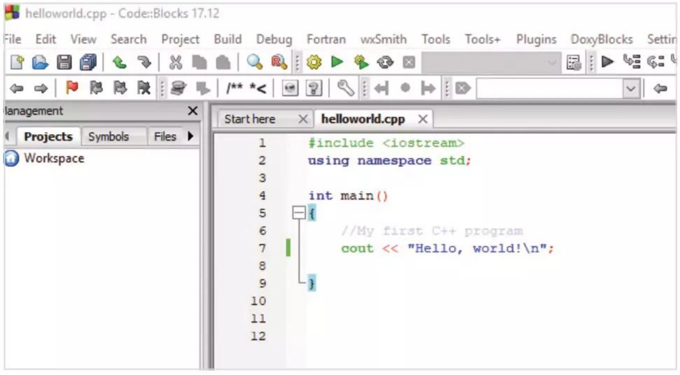
Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
<global>
main() : int
Start here
helloworld.cpp
C:\Users\david\Documents\C++\helloworld.exe

```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout << "Hello, world!\n";
}
```

```
Hello, world!

Process returned 0 (0x0)   execution time : 0.041 s
Press any key to continue.
```

STEP 7

Just as we mentioned in the previous pages, you don't need to have std::cout if you already declare using namespace std; at the beginning of the code. We could have easily clicked the Build/Run icon to begin with but it's worth going through the available options. You can also see that by building and running, the file has been saved.



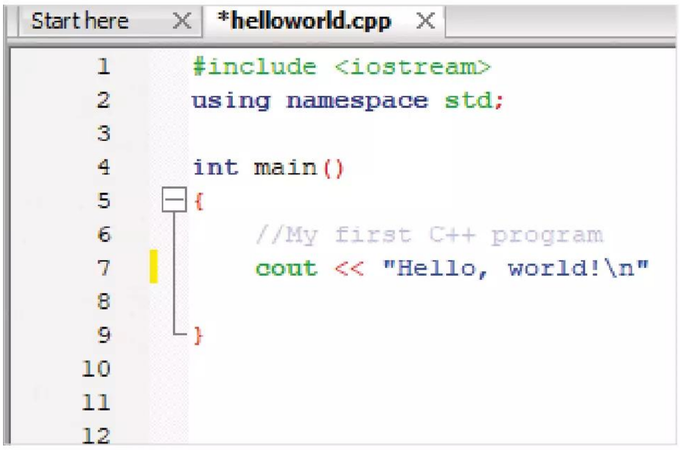
helloworld.cpp - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settin
Management
Projects Symbols Files
Workspace
Start here
helloworld.cpp

```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout << "Hello, world!\n";
}
```

STEP 8

Create a deliberate error in the code. Remove the semicolon from the cout line, so it reads:

```
cout << “Hello, world!\n”
```

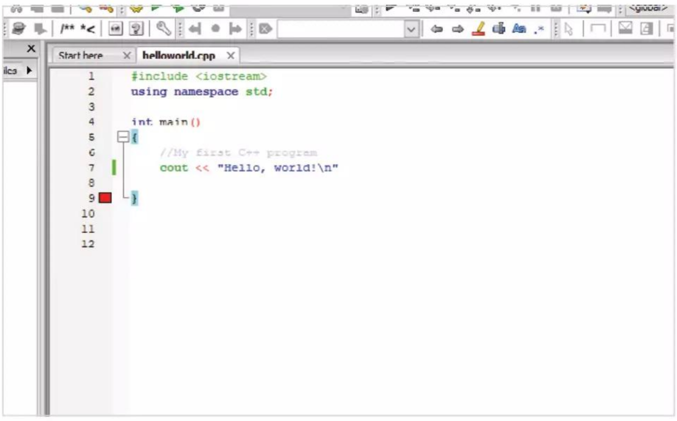


Start here
*helloworld.cpp

```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout << "Hello, world!\n"
}
```

STEP 9

Now click the Build and Run icon again to apply the changes to the code. This time Code::Blocks refuses to execute the code, due to the error you put in. In the Log pane at the bottom of the screen you are informed of the error, in this case: Expected ';' before '}' token, indicating the missing semicolon.



<global>
Start here
helloworld.cpp

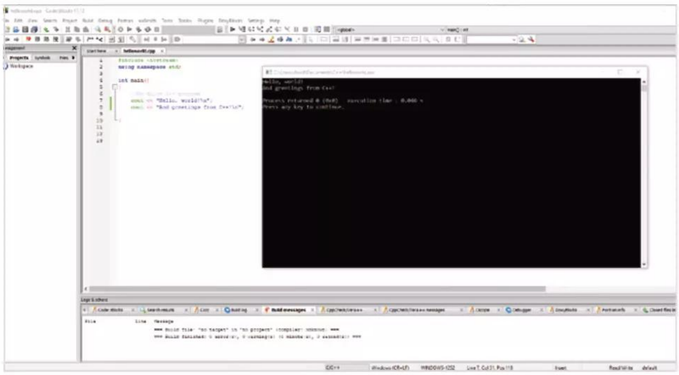
```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout << "Hello, world!\n"
}
```

STEP 10

Replace the semicolon and under the cout line, enter a new line to your code:

```
cout << “And greetings from C++!\n”;
```

The \n simply adds a new line under the last line of outputted text. Build and Run the code, to display your handiwork.



Using Comments

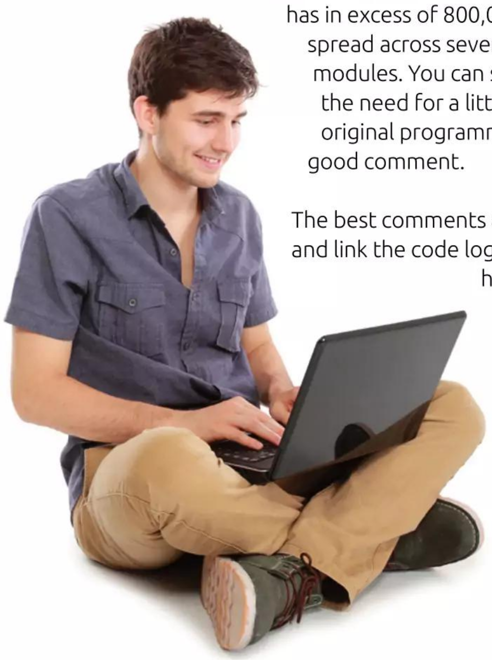
While comments may seem like a minor element to the many lines of code that combine to make a game, application or even an entire operating system, in actual fact they're probably one of the most important factors.

THE IMPORTANCE OF COMMENTING

Comments inside code are basically human readable descriptions that detail what the code is doing at that particular point. They don't sound especially important but code without comments is one of the many frustrating areas of programming, regardless of whether you're a professional or just starting out.

In short, all code should be commented in such a manner as to effectively describe the purpose of a line, section, or individual elements. You should get in to the habit of commenting as much as possible, by imagining that someone who doesn't know anything about programming can pick up your code and understand what it's going to do simply by reading your comments.

In a professional environment, comments are vital to the success of the code and ultimately, the company. In an organisation, many programmers work in teams alongside engineers, other developers, hardware analysts and so on. If you're a part of the team that's writing a bespoke piece of software for the company, then your comments help save a lot of time should something go wrong, and another team member has to pick up and follow the trail to pinpoint the issue.

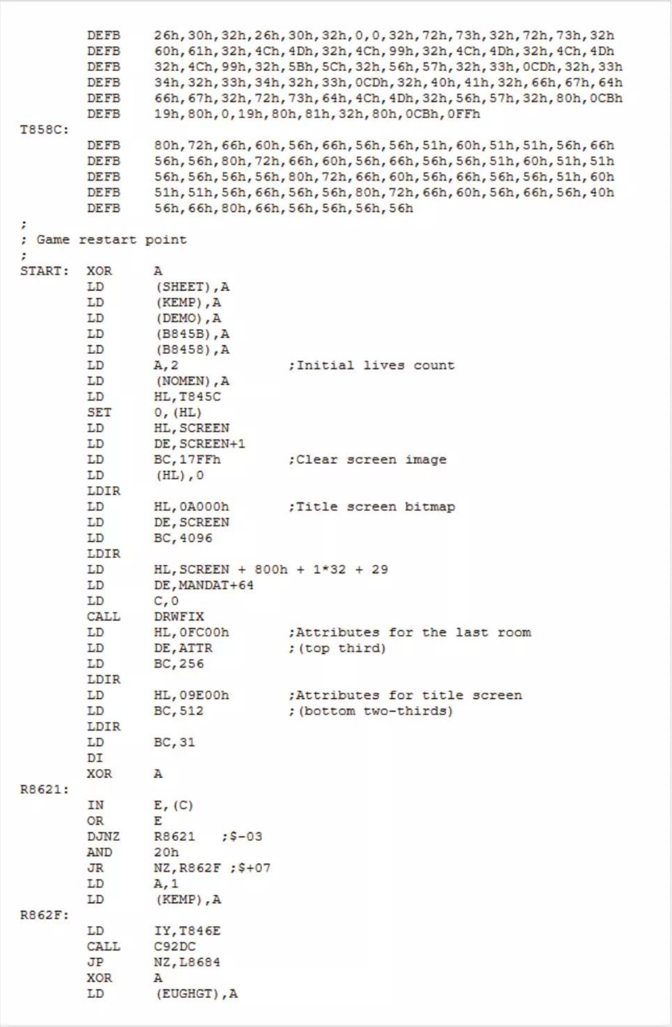


Place yourself in the shoes of someone whose job it is to find out what's wrong with a program. The program has in excess of 800,000 lines of code, spread across several different modules. You can soon appreciate the need for a little help from the original programmers in the form of a good comment.

The best comments are always concise and link the code logically, detailing what happens when the program hits this line or section. You don't need to comment on every line. Something along the lines of: if x==0 doesn't require you to comment that if x equals zero then do something; that's going to be obvious to the reader. However, if x equalling zero is something that drastically changes the program for the user, such as, they've run out of lives, then it certainly needs to commented on.

Even if the code is your own, you should write comments as if you were going to publicly share it with others. This way you can return to that code and always understand what it was you did or where it was you went wrong or what worked brilliantly.

Comments are good practise and once you understand how to add a comment where needed, you soon do it as if it's second nature.



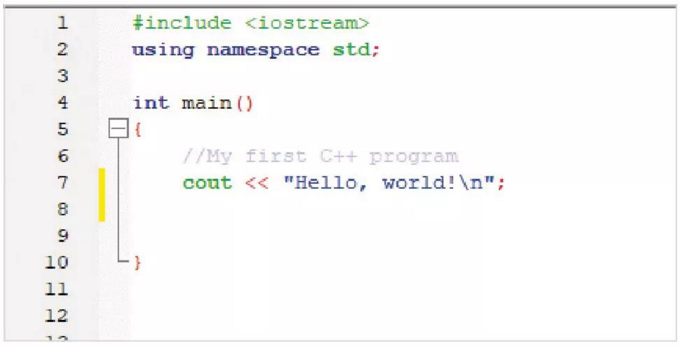
```
        DEFB    26h,30h,32h,26h,30h,32h,0,0,32h,72h,73h,32h,72h,73h,32h
        DEFB    60h,61h,32h,4Ch,4Dh,32h,4Ch,99h,32h,4Ch,4Dh,32h,4Ch,4Dh
        DEFB    32h,4Ch,99h,32h,5Bh,5Ch,32h,56h,57h,32h,33h,0CDh,32h,33h
        DEFB    34h,32h,33h,34h,32h,33h,0CDh,32h,40h,41h,32h,66h,67h,64h
        DEFB    66h,67h,32h,72h,73h,64h,4Ch,4Dh,32h,56h,57h,32h,80h,0CBh
        DEFB    19h,80h,0,19h,80h,81h,32h,80h,0CBh,0FFh
T858C:
        DEFB    80h,72h,66h,60h,56h,66h,56h,56h,51h,60h,51h,51h,56h,66h
        DEFB    56h,56h,80h,72h,66h,60h,56h,66h,56h,56h,51h,60h,51h,51h
        DEFB    56h,56h,56h,56h,80h,72h,66h,60h,56h,66h,56h,56h,51h,60h
        DEFB    51h,51h,56h,66h,56h,56h,80h,72h,66h,60h,56h,66h,56h,40h
        DEFB    56h,66h,80h,66h,56h,56h,56h,56h
;
; Game restart point
;
START:  XOR     A
        LD      (SHEET),A
        LD      (KEMP),A
        LD      (DEMO),A
        LD      (B845B),A
        LD      (B8458),A
        LD      A,2             ;Initial lives count
        LD      (NOMEN),A
        LD      HL,T845C
        SET     0,(HL)
        LD      HL,SCREEN
        LD      DE,SCREEN+1
        LD      BC,17FFh        ;Clear screen image
        LD      (HL),0
        LDIR
        LD      HL,0A000h       ;Title screen bitmap
        LD      DE,SCREEN
        LD      BC,4096
        LDIR
        LD      HL,SCREEN + 800h + 1*32 + 29
        LD      DE,MANDAT+64
        LD      C,0
        CALL    DRWFIX
        LD      HL,0FC00h       ;Attributes for the last room
        LD      DE,ATTR         ;(top third)
        LD      BC,256
        LDIR
        LD      HL,09E00h       ;Attributes for title screen
        LD      BC,512          ;(bottom two-thirds)
        LDIR
        LD      BC,31
        DI
        XOR     A
R8621:
        IN      E,(C)
        OR      E
        DJNZ    R8621   ;$-03
        AND     20h
        JR      NZ,R862F ;$+07
        LD      A,1
        LD      (KEMP),A
R862F:
        LD      IY,T846E
        CALL    C92DC
        JP      NZ,L8684
        XOR     A
        LD      (EUGHGT),A
```

C++ COMMENTS

Commenting in C++ involves using a double forward slash '/', or a forward slash and an asterisk, '/*'. You've already seen some brief examples but this is how they work.

STEP 1 Using the Hello World code as an example, you can easily comment on different sections of the code using the double forward slash:

```
//My first C++ program
cout << "Hello, world!\n";
```

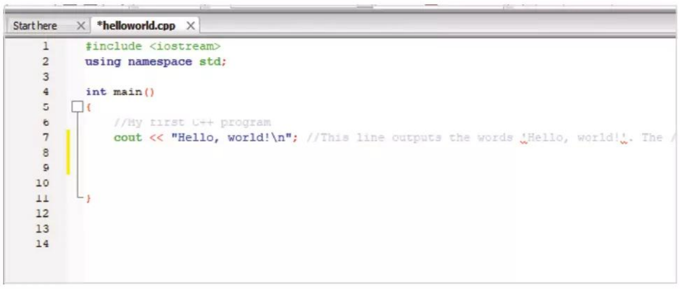


```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout << "Hello, world!\n";


}
```

STEP 2 However, you can also add comments to the end of a line of code, to describe in a better way what's going on:

```
cout << "Hello, world!\n"; //This line outputs the
words 'Hello, world!'. The \n denotes a new line.
```

Note, you don't have to put a semicolon at the end of a comment. This is because it's a line in the code that's ignored by the compiler.

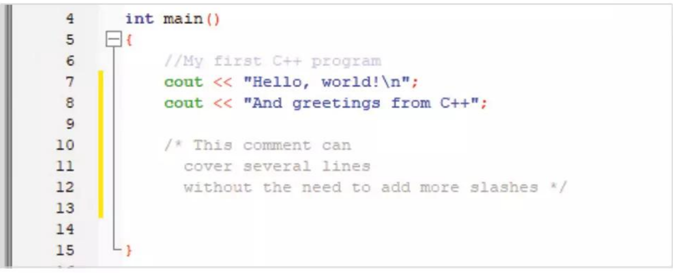


```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout << "Hello, world!\n"; //This line outputs the words 'Hello, world!'. The /


}
```

STEP 3 You can comment out several lines by using the forward slash and asterisk:

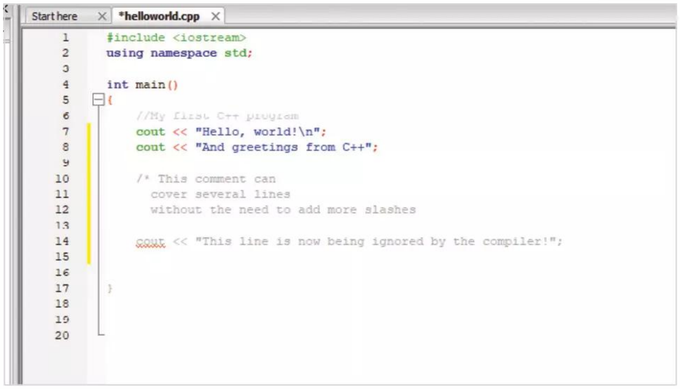
```
/* This comment can
      cover several lines
      without the need to add more slashes */
```

Just remember to finish the block comment with the opposite asterisk and forward slash.



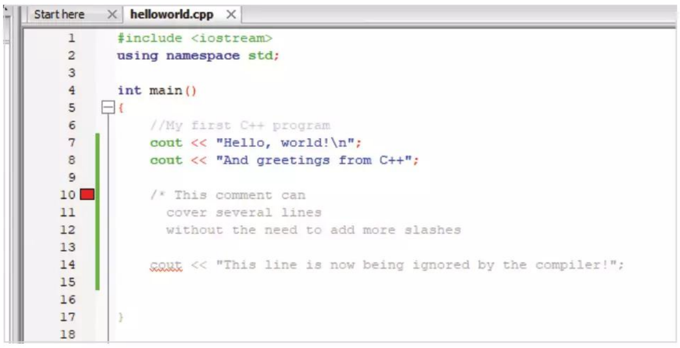
```
int main()
{
    //My first C++ program
    cout << "Hello, world!\n";
    cout << "And greetings from C++";

    /* This comment can
      cover several lines
      without the need to add more slashes */


}
```

STEP 4 Be careful when commenting, especially with block comments. It's very easy to forget to add the closing asterisk and forward slash and thus negate any code that falls inside the comment block.



```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout << "Hello, world!\n";
    cout << "And greetings from C++";

    /* This comment can
      cover several lines
      without the need to add more slashes

    cout << "This line is now being ignored by the compiler!";


}
```

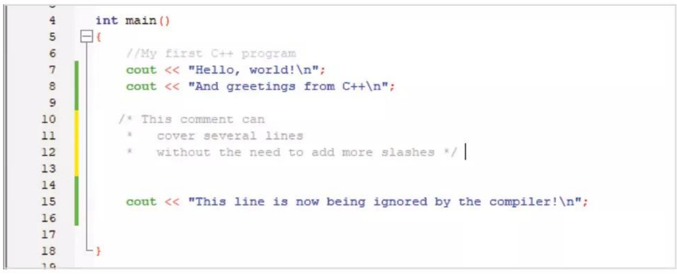
STEP 5 Obviously if you try and build and execute the code it errors out, complaining of a missing curly bracket '}' to finish off the block of code. If you've made the error a few times, then it can be time consuming to go back and rectify. Thankfully, the colour coding in Code::Blocks helps identify comments from code.



```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout << "Hello, world!\n";
    cout << "And greetings from C++";

    /* This comment can
      cover several lines
      without the need to add more slashes

    cout << "This line is now being ignored by the compiler!";


}
```

STEP 6 If you're using block comments, it's good practise in C++ to add an asterisk to each new line of the comment block. This also helps you to remember to close the comment block off before continuing with the code:

```
/* This comment can
  *   cover several lines
  *   without the need to add more slashes */
```



```
int main()
{
    //My first C++ program
    cout << "Hello, world!\n";
    cout << "And greetings from C++\n";

   /* This comment can
    *   cover several lines
    *   without the need to add more slashes */


    cout << "This line is now being ignored by the compiler!\n";


}
```

Variables

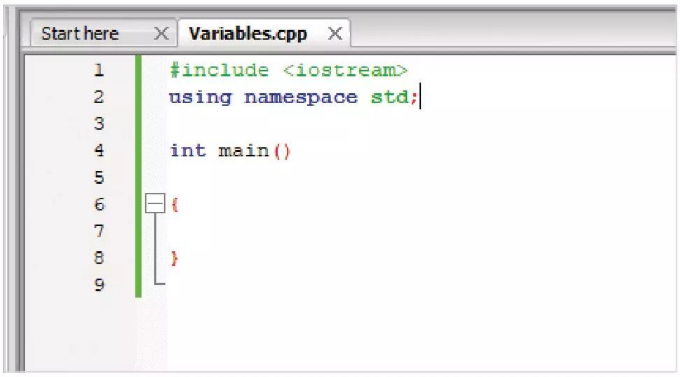
Variables differ slightly when using C++ as opposed to Python. In Python, you can simply state that ‘a’ equals 10 and a variable is assigned. However, in C++ a variable has to be declared with its type before it can be used.

THE DECLARATION OF VARIABLES

You can declare a C++ variable by using statements within the code. There are several distinct types of variables you can declare. Here’s how it works.

STEP 1 Open up a new, blank C++ file and enter the usual code headers:

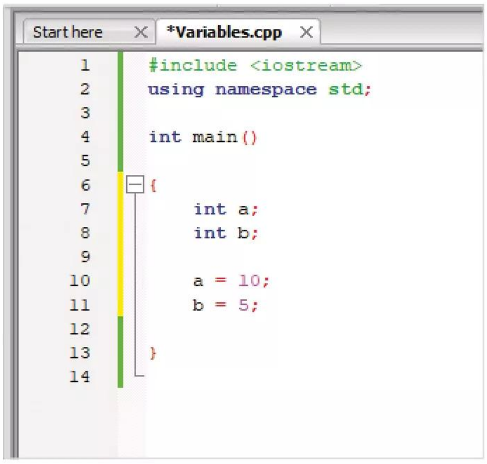
```
#include <iostream>
using namespace std;

int main()
{

}
```



```
#include <iostream>
using namespace std;

int main()

{

}

```

STEP 2 Start simple by creating two variables, a and b, with one having a value of 10 and the other 5. You can use the data type int to declare these variables. Within the curly brackets, enter:

```
int a;
int b;

a = 10;
b = 5;
```

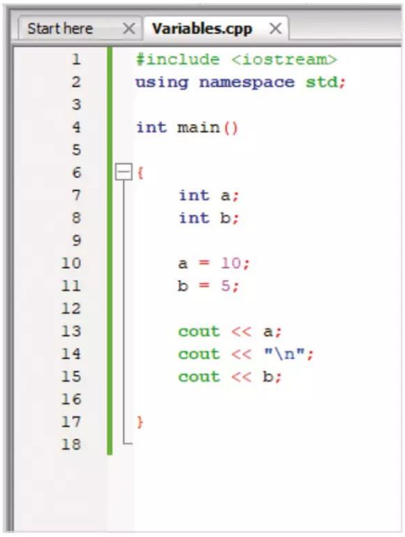


```
#include <iostream>
using namespace std;

int main()

{
    int a;
    int b;

    a = 10;
    b = 5;

}

```

STEP 3 You can build and run the code but it won’t do much, other than store the values 10 and 5 to the integers a and b. To output the contents of the variables, add:

```
cout << a;
cout << ”\n”;
cout << b;
```

The cout << “\n”; part simply places a new line between the output of 10 and 5.

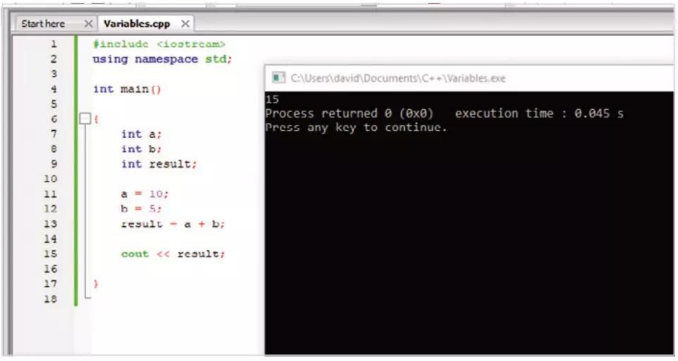


```
#include <iostream>
using namespace std;

int main()

{
    int a;
    int b;

    a = 10;
    b = 5;

    cout << a;
    cout << "\n";
    cout << b;

}

```

STEP 4 Naturally you can declare a new variable, call it result and output some simple arithmetic:

```
int result;

result = a + b;
cout << result;
```

Insert the above into the code as per the screenshot.

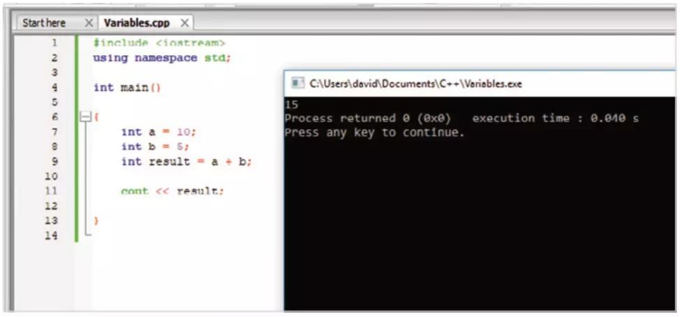


```
#include <iostream>
using namespace std;

int main()

{
    int a;
    int b;
    int result;

    a = 10;
    b = 5;
    result = a + b;

    cout << result;

}

```

```
15
Process returned 0 (0x0)   execution time : 0.045 s
Press any key to continue.
```

STEP 5 You can assign a value to a variable as soon as you declare it. The code you've typed in could look like this, instead:

```
int a = 10;
int b = 5;
int result = a + b;

cout << result;
```



```
#include <iostream>
using namespace std;

int main()

{

    int a = 10;
    int b = 5;
    int result = a + b;

    cout << result;

}
```

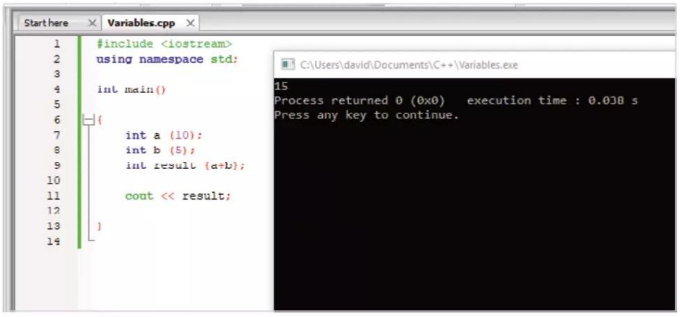
```
15
Process returned 0 (0x0)   execution time : 0.040 s
Press any key to continue.
```

STEP 6 Specific to C++, you can also use the following to assign values to a variable as soon as you declare them:

```
int a (10);
int b (5);
```

Then, from the C++ 2011 standard, using curly brackets:

```
int result {a+b};
```

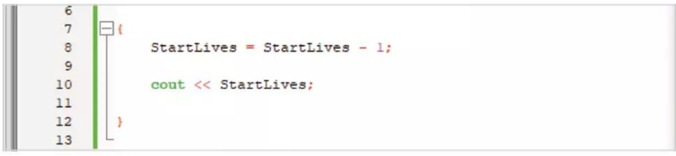


```
#include <iostream>
using namespace std;

int main()

{

    int a (10);
    int b (5);
    int result {a+b};

    cout << result;

}
```

```
15
Process returned 0 (0x0)   execution time : 0.038 s
Press any key to continue.
```

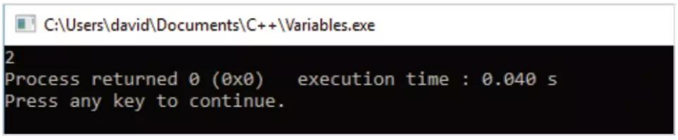
STEP 7 You can create global variables, which are variables that are declared outside any function and used in any function within the entire code. What you've used so far are local variables: variables used inside the function. For example:

```
#include <iostream>
using namespace std;
int StartLives = 3;

int main ()

{
      startLives = StartLives - 1;
      cout << StartLives;
}
```



```
{

    StartLives = StartLives - 1;

    cout << StartLives;

}
```

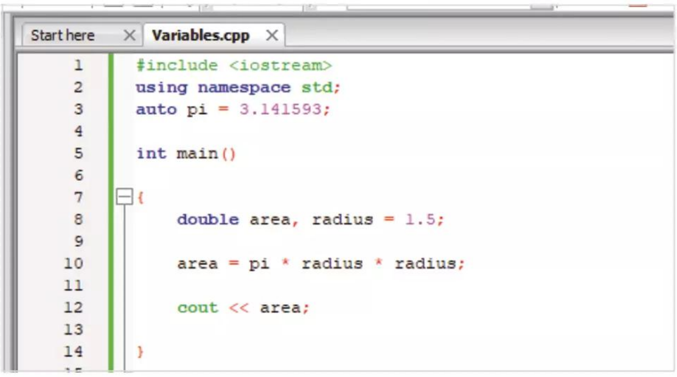
STEP 8 The previous step creates the variable StartLives, which is a global variable. In a game, for example, a player's lives go up or down depending on how well or how bad they're doing. When the player restarts the game, the StartLives returns to its default state: 3. Here we've assigned 3 lives, then subtracted 1, leaving 2 lives left.



```
2
Process returned 0 (0x0)   execution time : 0.040 s
Press any key to continue.
```

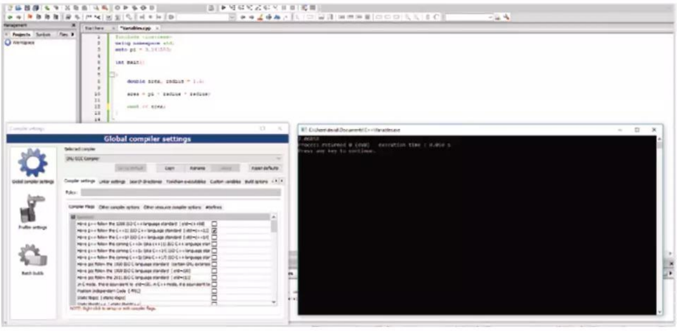
STEP 9 The modern C++ compiler is far more intelligent than most programmers give it credit. While there are numerous data types you can declare for variables, you can in fact use the auto feature:

```
#include <iostream>
using namespace std;
auto pi = 3.141593;

int main()

{
    double area, radius = 1.5;

    area = pi * radius * radius;

    cout << area;

}
```



```
#include <iostream>
using namespace std;
auto pi = 3.141593;

int main()

{

    double area, radius = 1.5;

    area = pi * radius * radius;

    cout << area;

}
```

STEP 10 A couple of new elements here: first, auto won't work unless you go to Settings > Compiler and tick the box labelled 'Have G++ follow the C++11 ISO C++ Language Standard [-std=c++1]'. Then, the new data type, double, which means double-precision floating point value. Enable C++11, then build and run the code. The result should be 7.06858.



Data Types

Variables, as we've seen, store information that the programmer can then later call up, and manipulate if required. Variables are simply reserved memory locations that store the values the programmer assigns, depending on the data type used.

THE VALUE OF DATA

There are many different data types available for the programmer in C++, such as an integer, floating point, Boolean, character and so on. It's widely accepted that there are seven basic data types, often called Primitive Built-in Types; however, you can create your own data types should the need ever arise within your code.

The seven basic data types are:

TYPE	COMMAND
Integer	`Integer`
Floating Point	`float`
Character	`char`
Boolean	`bool`
Double Floating Point	`double`
Wide Character	`wchar_t`
No Value	`void`

These basic types can also be extended using the following modifiers: Long, Short, Signed and Unsigned. Basically this means the modifiers can expand the minimum and maximum range values for each data type. For example, the int data type has a default value range of -2147483648 to 2147483647, a fair value, you would agree.

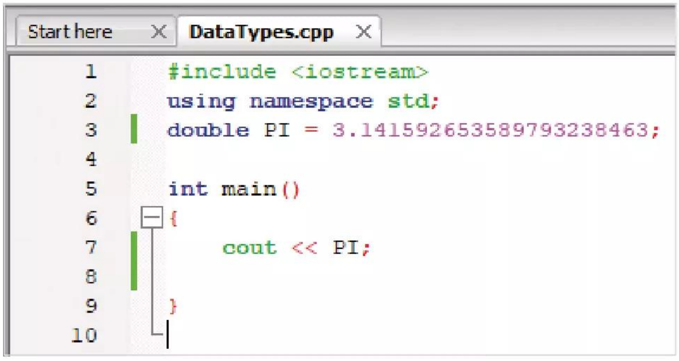
Now, if you were to use one of the modifiers, the range alters:

Unsigned int = 0 to 4294967295
Signed int = -2147483648 to 2147483647
Short int = -32768 to 32767
Unsigned Short int = 0 to 65,535
Signed Short int = -32768 to 32767
Long int = -2147483647 to 2147483647
Signed Long int = -2147483647 to 2147483647
Unsigned Long int = 0 to 4294967295

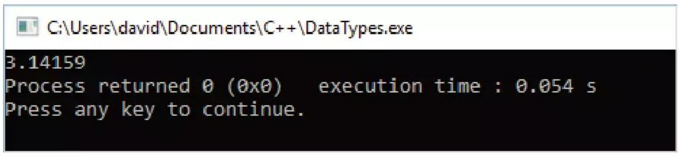
Naturally you can get away with using the basic type without the modifier, as there's plenty of range provided with each data type. However, it's considered good C++ programming practise to use the modifiers when possible.

There are issues when using the modifiers though. Double represents a double-floating point value, which you can use for incredibly accurate numbers but those numbers are only accurate up to the fifteenth decimal place. There's also the problem when displaying such numbers in C++ using the cout function, in that cout by default only outputs the first five decimal places. You can combat that by adding a cout.precision () function and adding a value inside the brackets, but even then you're still limited by the accuracy of the double data type. For example, try this code:

```
#include <iostream>
using namespace std;
double PI = 3.141592653589793238463;

int main()
{
    cout << PI;
}
```



```
#include <iostream>
using namespace std;
double PI = 3.141592653589793238463;

int main()
{
    cout << PI;

}

```

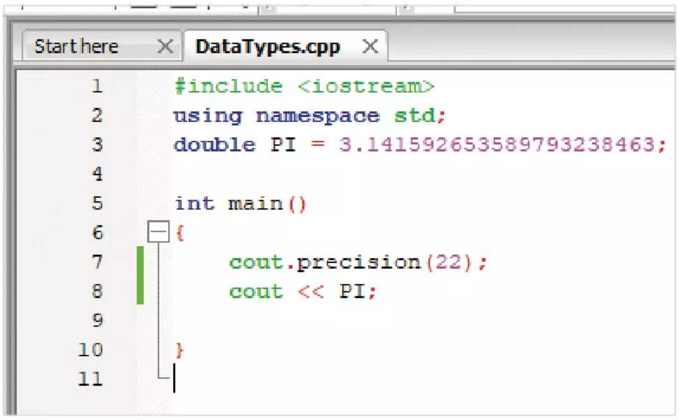


```
3.14159
Process returned 0 (0x0)   execution time : 0.054 s
Press any key to continue.
```

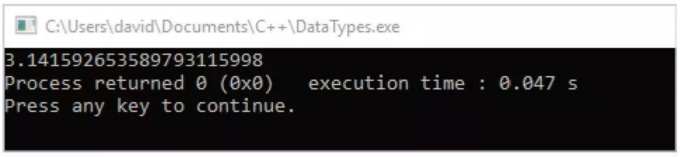
Build and run the code and as you can see the output is only 3.14159, representing cout's limitations in this example.

You can alter the code including the aforementioned cout.precision function, for greater accuracy. Take precision all the way up to 22 decimal places, with the following code:

```
#include <iostream>
using namespace std;
double PI = 3.141592653589793238463;

int main()
{
    cout.precision(22);
    cout << PI;

}
```



```
#include <iostream>
using namespace std;
double PI = 3.14159265358979323846З;

int main()
{
    cout.precision(22);
    cout << PI;

}
```



```
3.141592653589793115998
Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

Again, build and run the code; as you can see from the command line window, the number represented by the variable PI is different to the number you've told C++ to use in the variable. The output reads the value of PI as 3.141592653589793115998, with the numbers going awry from the fifteenth decimal place.

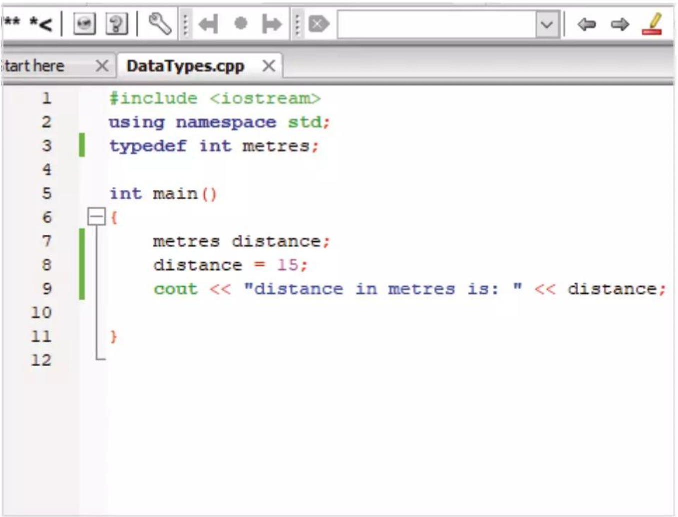
Calculator
Scientific
DEG
HYP
F-E
MC
MR
M+
M-
MS
M˅
x²
xʸ
sin
cos
tan
√
10ˣ
log
Exp
Mod
↑
CE
C
⌫
÷
π
7
8
9
×
n!
4
5
6
−
±
1
2
3
+
(
)
0
.
=

15.142857142857142857142857142857

This is mainly due to the conversion from binary in the compiler and that the IEEE 754 double precision standard occupies 64-bits of data, of which 52-bits are dedicated to the significant (the significant digits in a floating-point number) and roughly 3.5-bits are taken holding the values 0 to 9. If you divide 53 by 3.5, then you arrive at 15.142857 recurring, which is 15-digits of precision.

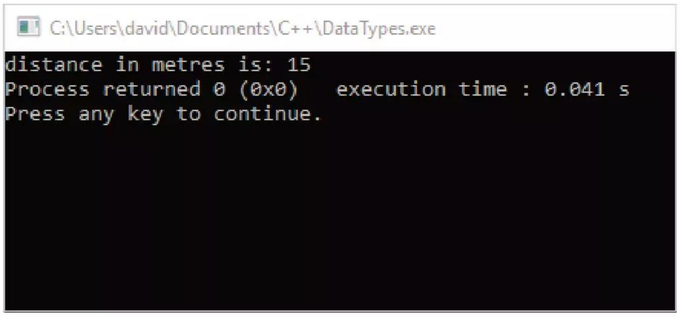
To be honest, if you're creating code that needs to be accurate to more than fifteen decimal places, then you wouldn't be using C++, you would use some scientific specific language with C++ as the connective tissue between the two languages.

You can create your own data types, using an alias-like system called typedef. For example:



```
#include <iostream>
using namespace std;
typedef int metres;

int main()
{
    metres distance;
    distance = 15;
    cout << "distance in metres is: " << distance;

}
```

```
#include <iostream>
using namespace std;
typedef int metres;

int main()
{
    metres distance;
    distance = 15;
    cout << “distance in metres is: “ << distance;

}
```



```
distance in metres is: 15
Process returned 0 (0x0)   execution time : 0.041 s
Press any key to continue.
```

This code when executed creates a new int data type called metres. Then, in the main code block, there's a new variable called distance, which is an integer; so you're basically telling the compiler that there's another name for int. We assigned the value 15 to distance and displayed the output: distance in metres is 15.

It might sound a little confusing to begin with but the more you use C++ and create your own code, the easier it becomes.

Strings

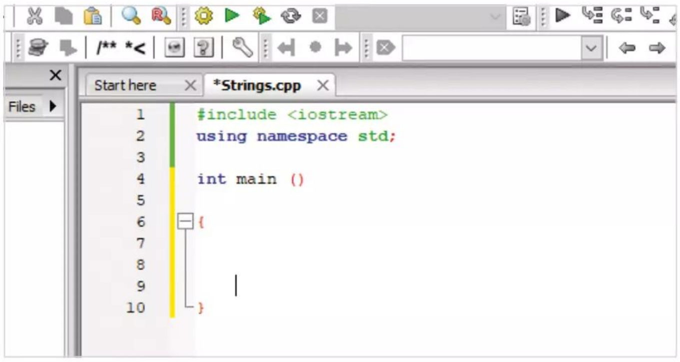
Strings are objects that represent and hold sequences of characters. For example, you could have a universal greeting in your code ‘Welcome’ and assign that as a string to be called up wherever you like in the program.

STRING THEORY

There are different ways in which you can create a string of characters, which historically are all carried over from the original C language, and are still supported by C++.

STEP 1 To create a string you use the char function. Open a new C++ file and begin with the usual header:

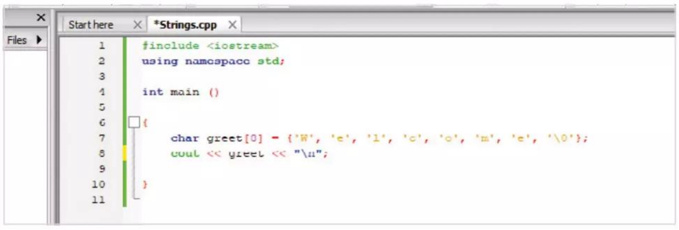
```
#include <iostream>
using namespace std;

int main ()

{

}
```



```
#include <iostream>
using namespace std;

int main ()

{


}
```

STEP 2 It’s easy to confuse a string with an array. Here’s an array, which can be terminated with a null character:

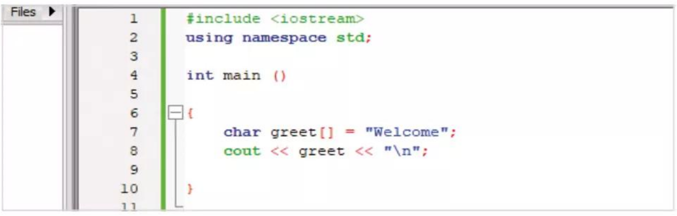
```
#include <iostream>
using namespace std;

int main ()

{
    char greet[8] = {‘W’, ‘e’, ‘l’, ‘c’, ‘o’, ‘m’, ‘e’, ‘\0’};
    cout << greet << “\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()

{
    char greet[8] = {'W', 'e', 'l', 'c', 'o', 'm', 'e', '\0'};
    cout << greet << "\n";

}
```

STEP 3 Build and run the code, and ‘Welcome’ appears on the screen. While this is perfectly fine, it’s not a string. A string is a class, which defines objects that can be represented as a stream of characters and doesn’t need to be terminated like an array. The code can therefore be represented as:

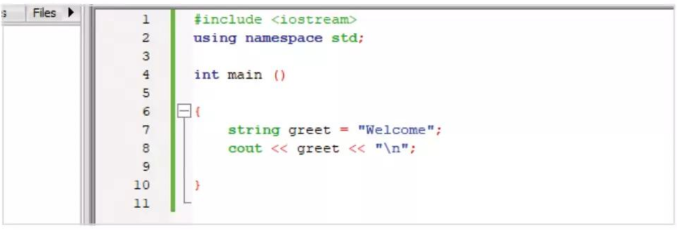
```
#include <iostream>
using namespace std;

int main ()

{
    char greet[] = “Welcome”;
    cout << greet << “\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()

{
    char greet[] = "Welcome";
    cout << greet << "\n";

}
```

STEP 4 In C++ there’s also a string function, which works in much the same way. Using the greeting code again, you can enter:

```
#include <iostream>
using namespace std;

int main ()

{
    string greet = “Welcome”;
    cout << greet << “\n”;

}
```

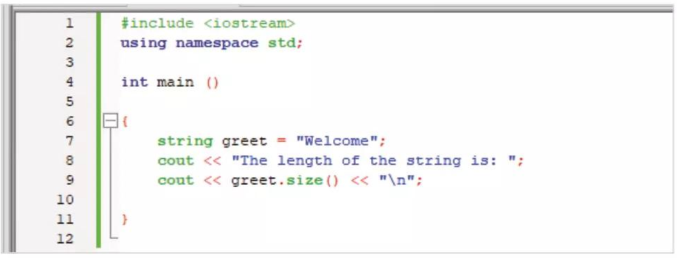


```
#include <iostream>
using namespace std;

int main ()

{
    string greet = "Welcome";
    cout << greet << "\n";

}
```

STEP 5

There are also many different operations that you can apply with the string function. For instance, to get the length of a string you can use:

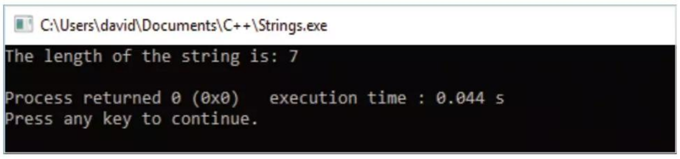
```
#include <iostream>
using namespace std;

int main ()

{
    string greet = “Welcome”;
    cout << “The length of the string is: “;
    cout << greet.size() << “\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()

{

    string greet = "Welcome";
    cout << "The length of the string is: ";
    cout << greet.size() << "\n";

}
```

STEP 6

You can see that we used greet.size() to output the length, the number of characters there are, of the contents of the string. Naturally, if you call your string something other than greet, then you need to change the command to reflect this. It's always stringname.operation. Build and run the code to see the results.



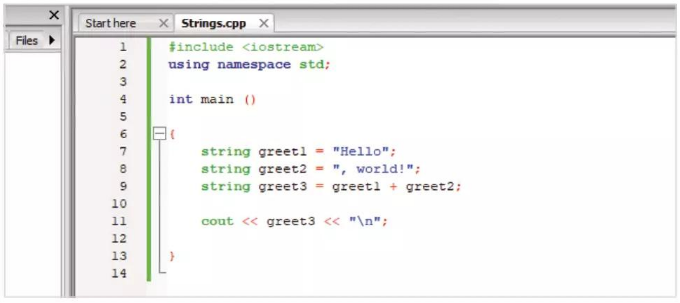
C:\Users\david\Documents\C++\Strings.exe

```
The length of the string is: 7

Process returned 0 (0x0)   execution time : 0.044 s
Press any key to continue.
```

STEP 7

You can of course add strings together, or rather combine them to form longer strings:

```
#include <iostream>
using namespace std;

int main ()

{
    string greet1 = “Hello”;
    string greet2 = “, world!”;
    string greet3 = greet1 + greet2;

    cout << greet3 << “\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()

{

    string greet1 = "Hello";
    string greet2 = ", world!";
    string greet3 = greet1 + greet2;

    cout << greet3 << "\n";

}
```

STEP 8

Just as you might expect, you can mix in an integer and store something to do with the string. In this example, we created int length, which stores the result of string.size() and outputs it to the user:

```
#include <iostream>
using namespace std;

int main ()

{
    int length;
    string greet1 = “Hello”;
    string greet2 = “, world!”;
    string greet3 = greet1 + greet2;

    length = greet3.size();

    cout << “The length of the combined strings
is: “ << length << “\n”;

}
```

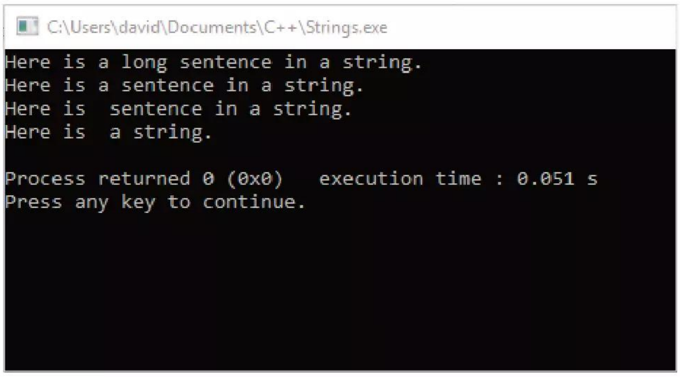
STEP 9

Using the available operations that come with the string function, you can manipulate the contents of a string. For example, to remove characters from a string you could use:

```
#include <iostream>
using namespace std;

int main ()

{
  string strg (“Here is a long sentence in a
string.”);
  cout << strg << ‘\n’;

  strg.erase (10,5);
  cout << strg << ‘\n’;

  strg.erase (strg.begin()+8);
  cout << strg << ‘\n’;

  strg.erase (strg.begin()+9, strg.end()-9);
  cout << strg << ‘\n’;

}
```

STEP 10

It's worth spending some time playing around with the numbers, which are the character positions in the string. Occasionally, it can be hit and miss whether you get it right, so practice makes perfect. Take a look at the screenshot to see the result of the code.



C:\Users\david\Documents\C++\Strings.exe

```
Here is a long sentence in a string.
Here is a sentence in a string.
Here is  sentence in a string.
Here is  a string.

Process returned 0 (0x0)   execution time : 0.051 s
Press any key to continue.
```

C++ Maths

Programming is mathematical in nature and as you might expect, there's plenty of built-in scope for some quite intense maths. C++ has a lot to offer someone who's implementing mathematical models into their code. It can be extremely complex or relatively simple.

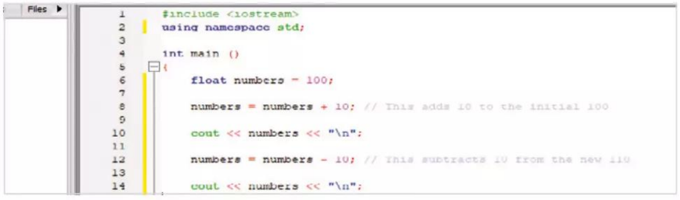
C++ = MC²

The basic mathematical symbols apply in C++ as they do in most other programming languages. However, by using the C++ Math Library, you can also calculate square roots, powers, trig and more.

STEP 1

C++'s mathematical operations follow the same patterns as those taught in school, in that multiplication and division take precedence over addition and subtraction. You can alter that though. For now, create a new file and enter:

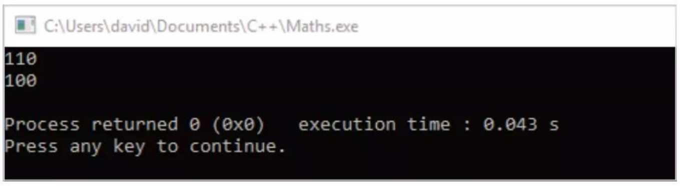
```
#include <iostream>
using namespace std;

int main ()
{
    float numbers = 100;

    numbers = numbers + 10; // This adds 10 to the
initial 100

    cout << numbers << “\n”;

    numbers = numbers - 10; // This subtracts 10
from the new 110

    cout << numbers << “\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()
{
     float numbers = 100;

     numbers = numbers + 10; // This adds 10 to the initial 100

     cout << numbers << "\n";

     numbers = numbers - 10; // This subtracts 10 from the new 110

     cout << numbers << "\n";
```

STEP 2

While simple, it does get the old maths muscle warmed up. Note that we used a float for the numbers variable. While you can happily use an integer, if you suddenly started to use decimals, you would need to change to a float or a double, depending on the accuracy needed. Run the code and see the results.

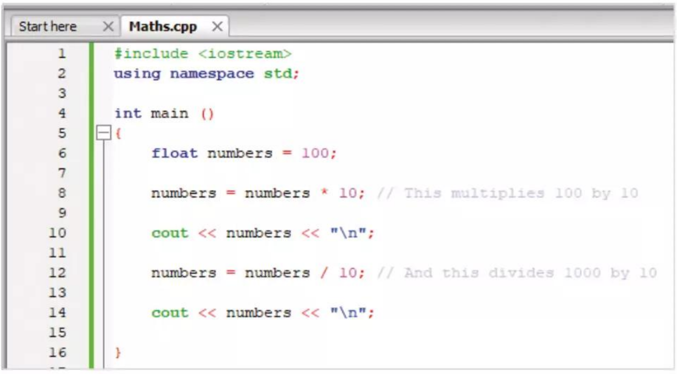


```
110
100

Process returned 0 (0x0)   execution time : 0.043 s
Press any key to continue.
```

STEP 3

Multiplication and division can be applied as such:

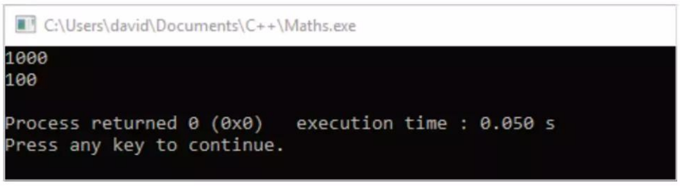
```
#include <iostream>
using namespace std;

int main ()
{
    float numbers = 100;

    numbers = numbers * 10; // This multiplies 100
by 10

    cout << numbers << “\n”;

    numbers = numbers / 10; // And this divides
1000 by 10

    cout << numbers << “\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()
{
     float numbers = 100;

     numbers = numbers * 10; // This multiplies 100 by 10

     cout << numbers << "\n";

     numbers = numbers / 10; // And this divides 1000 by 10

     cout << numbers << "\n";

}
```

STEP 4

Again, execute the simple code and see the results. While not particularly interesting, it's a start into C++ maths. We used a float here, so you can play around with the code and multiply by decimal places, as well as divide, add and subtract.

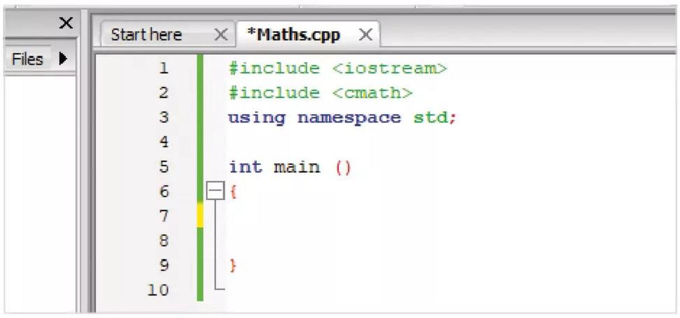


```
1000
100

Process returned 0 (0x0)   execution time : 0.050 s
Press any key to continue.
```

STEP 5

The interesting maths content comes when you call upon the C++ Math Library. Within this header are dozens of mathematical functions along with further operations. Everything from computing cosine to arc tangent with two parameters, to the value of PI. You can call the header with:

```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{

}
```

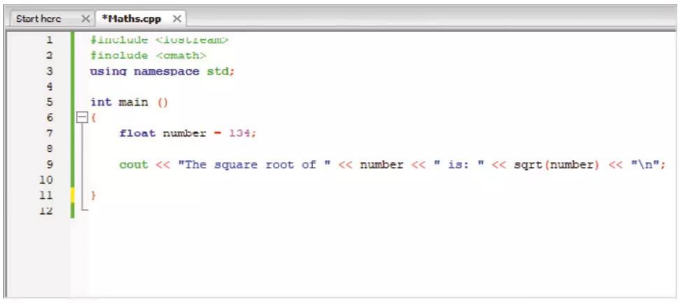


```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{


}
```

STEP 6

Start by getting the square root of a number:

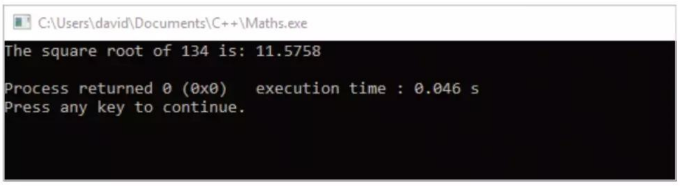
```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{
    float number = 134;

    cout << “The square root of “ << number << “
is: “ << sqrt(number) << “\n”;

}
```



```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{
    float number = 134;

    cout << "The square root of " << number << " is: " << sqrt(number) << "\n";

}
```

STEP 7

Here we created a new float called number and used the sqrt(number) function to display the square root of 134, the value of the variable, number. Build and run the code, and your answer reads 11.5758.

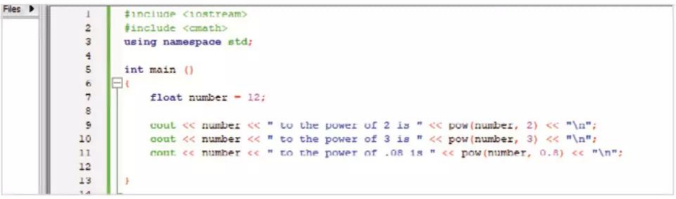


```
The square root of 134 is: 11.5758

Process returned 0 (0x0)   execution time : 0.046 s
Press any key to continue.
```

STEP 8

Calculating powers of numbers can be done with:

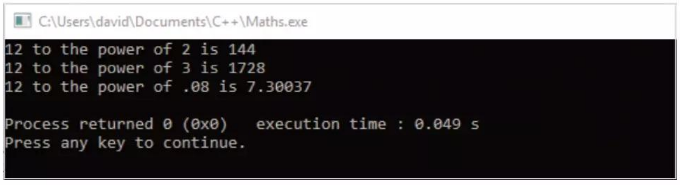
```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{
    float number = 12;

    cout << number << “ to the power of 2 is “ <<
pow(number, 2) << “\n”;
    cout << number << “ to the power of 3 is “ <<
pow(number, 3) << “\n”;
    cout << number << “ to the power of .08 is “
<< pow(number, 0.8) << “\n”;

}
```



```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{
    float number = 12;

    cout << number << " to the power of 2 is " << pow(number, 2) << "\n";
    cout << number << " to the power of 3 is " << pow(number, 3) << "\n";
    cout << number << " to the power of .08 is " << pow(number, 0.8) << "\n";

}
```

STEP 9

Here we created a float called number with the value of 12, and the pow(variable, power) is where the calculation happens. Of course, you can calculate powers and square roots without using variables. For example, pow (12, 2) outputs the same value as the first cout line in the code.

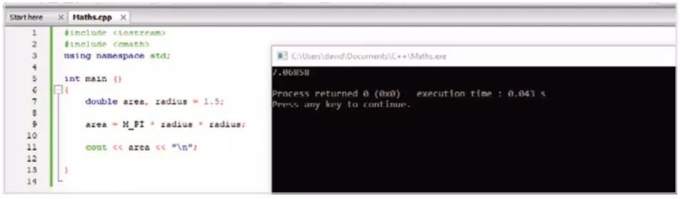


```
12 to the power of 2 is 144
12 to the power of 3 is 1728
12 to the power of .08 is 7.30037

Process returned 0 (0x0)   execution time : 0.049 s
Press any key to continue.
```

STEP 10

The value of Pi is also stored in the cmath header library. It can be called up with the M_PI function. Enter cout << M_PI; into the code and you get 3.14159; or you can use it to calculate:

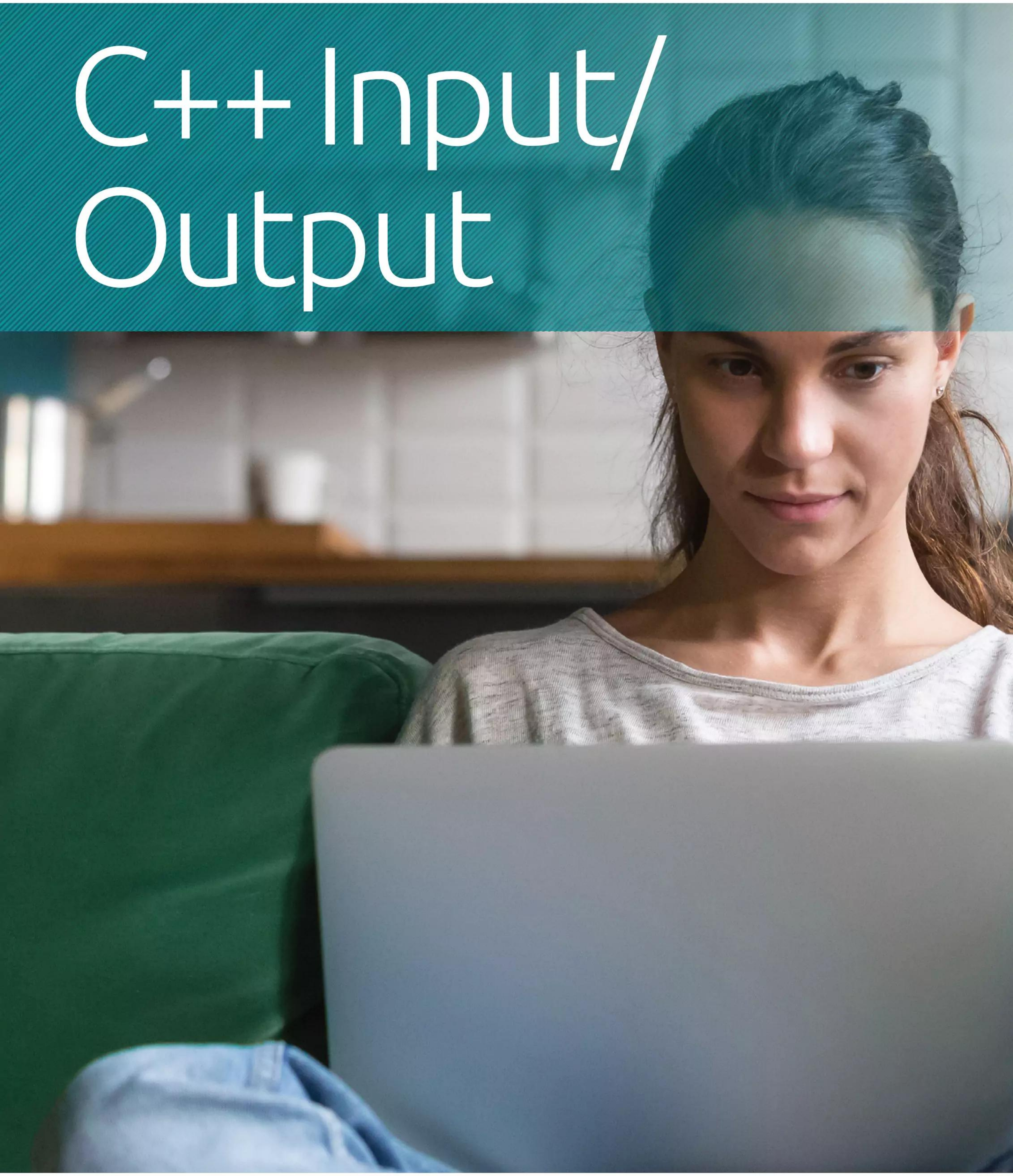
```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{
    double area, radius = 1.5;

    area = M_PI * radius * radius;

    cout << area << “\n”;

}
```



```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{
    double area, radius = 1.5;

    area = M_PI * radius * radius;

    cout << area << "\n";

}
```

```
7.06858

Process returned 0 (0x0)   execution time : 0.043 s
Press any key to continue.
```

C++ Input/ Output



There's a satisfying feeling when you program code that asks the user for input, then uses that input to produce something that the user can see. Even if it's simply asking for someone's name, and displaying a personal welcome message, it's a big leap forward.

User interaction, character literals, defining constants and file input and output are all covered in the following pages. All of which help you to understand how a C++ program works better.

116 User Interaction
118 Character Literals
120 Defining Constants
122 File Input/Output

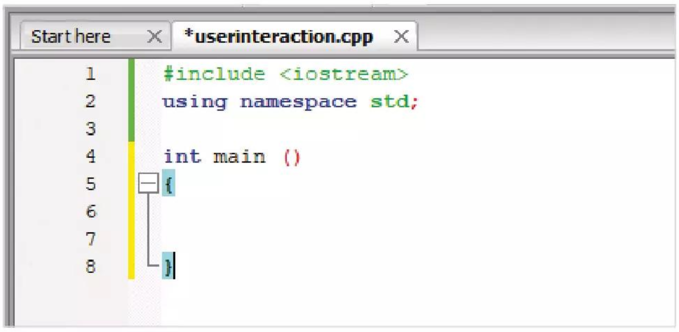
User Interaction

There’s nothing quite as satisfying as creating a program that responds to you. This basic user interaction is one of the most taught aspects of any language and with it you’re able to do much more than simply greet the user by name.

HELLO, DAVE

You have already used cout, the standard output stream, throughout our code. Now you’re going to be using cin, the standard input stream, to prompt a user response.

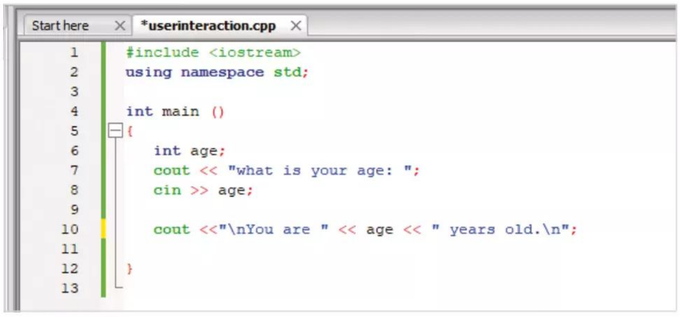
STEP 1 Anything that you want the user to input into the program needs to be stored somewhere in the system memory, so it can be retrieved and used. Therefore, any input must first be declared as a variable, so it’s ready to be used by the user. Start by creating a blank C++ file with headers.



```
#include <iostream>
using namespace std;

int main ()
{


}
```

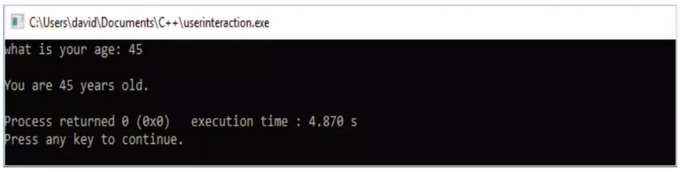
STEP 2 The data type of the variable must also match the type of input you want from the user. For example, to ask a user their age, you would use an integer like this:

```
#include <iostream>
using namespace std;

int main ()
{
   int age;
   cout << “what is your age: “;
   cin >> age;

   cout <<”\nYou are “ << age << “ years old.\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()
{
    int age;
    cout << "what is your age: ";
    cin >> age;

    cout <<"\nYou are " << age << " years old.\n";

}
```

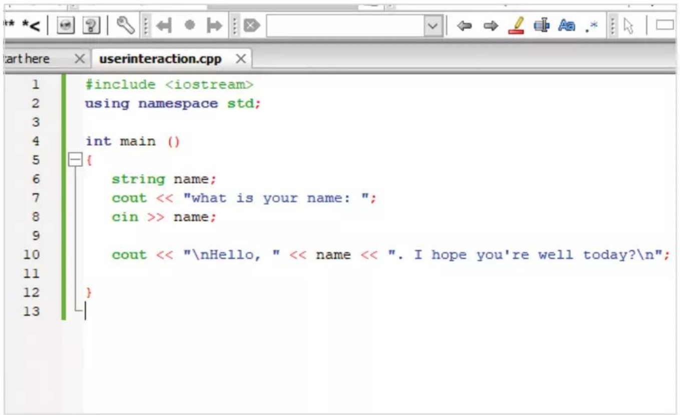
STEP 3 The cin command works in the opposite way from the cout command. With the first cout line you’re outputting ‘What is your age’ to the screen, as indicated with the chevrons. Cin uses opposite facing chevrons, indicating an input. The input is put into the integer age and called up in the second cout command. Build and run the code.



```
hat is your age: 45

ou are 45 years old.

rocess returned 0 (0x0)   execution time : 4.870 s
ress any key to continue.
```

STEP 4 If you’re asking a question, you need to store the input as a string; to ask the user their name, you would use:

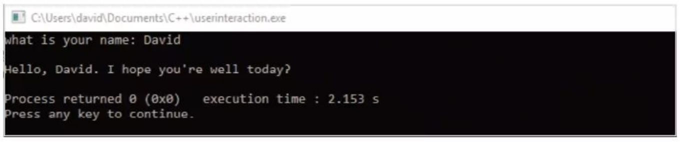
```
#include <iostream>
using namespace std;

int main ()
{
   string name;
   cout << “what is your name: “;
   cin >> name;

   cout << “\nHello, “ << name << “. I hope you’re
well today?\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()
{
    string name;
    cout << "what is your name: ";
    cin >> name;

    cout << "\nHello, " << name << ". I hope you're well today?\n";

}
```

STEP 5

The principal works the same as the previous code. The user's input, their name, is stored in a string, because it contains multiple characters, and retrieved in the second cout line. As long as the variable 'name' doesn't change, then you can recall it wherever you like in your code

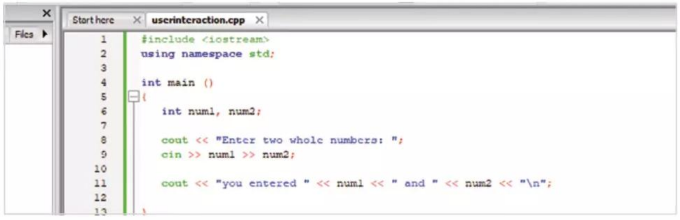


```
C:\Users\david\Documents\C++\userinteraction.exe
what is your name: David

Hello, David. I hope you're well today?

Process returned 0 (0x0)   execution time : 2.153 s
Press any key to continue.
```

STEP 6

You can chain input requests to the user but just make sure you have a valid variable to store the input to begin with. Let's assume you want the user to enter two whole numbers:

```
#include <iostream>
using namespace std;

int main ()
{
   int num1, num2;

   cout << “Enter two whole numbers: “;
   cin >> num1 >> num2;

   cout << “you entered “ << num1 << “ and “ <<
num2 << “\n”;

}
```

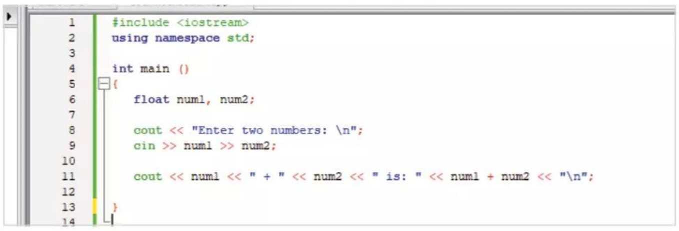


```
#include <iostream>
using namespace std;

int main ()
{
   int num1, num2;

   cout << "Enter two whole numbers: ";
   cin >> num1 >> num2;

   cout << "you entered " << num1 << " and " << num2 << "\n";
```

STEP 7

Likewise, inputted data can be manipulated once you have it stored in a variable. For instance, ask the user for two numbers and do some maths on them:

```
#include <iostream>
using namespace std;

int main ()
{
   float num1, num2;

   cout << “Enter two numbers: \n”;
   cin >> num1 >> num2;

   cout << num1 << “ + “ << num2 << “ is: “ <<
num1 + num2 << “\n”;

}
```

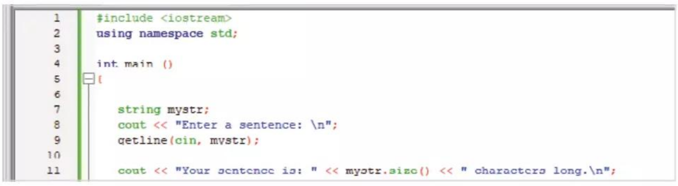


```
#include <iostream>
using namespace std;

int main ()
{
   float num1, num2;

   cout << "Enter two numbers: \n";
   cin >> num1 >> num2;

   cout << num1 << " + " << num2 << " is: " << num1 + num2 << "\n";

}
```

STEP 8

While cin works well for most input tasks, it does have a limitation. Cin always considers spaces as a terminator, so it's designed for just single words not multiple words. However, getline takes cin as the first argument and the variable as the second:

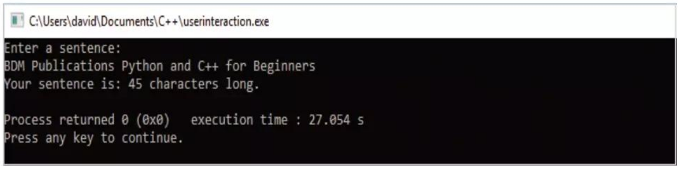
```
#include <iostream>
using namespace std;

int main ()
{

   string mystr;
   cout << “Enter a sentence: \n”;
   getline(cin, mystr);

   cout << “Your sentence is: “ << mystr.size() <<
“ characters long.\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()
{

   string mystr;
   cout << "Enter a sentence: \n";
   getline(cin, mystr);

   cout << "Your sentence is: " << mystr.size() << " characters long.\n";
```

STEP 9

Build and execute the code, then enter a sentence with spaces. When you're done the code reads the number of characters. If you remove the getline line and replace it with cin >> mystr and try again, the result displays the number of characters up to the first space.

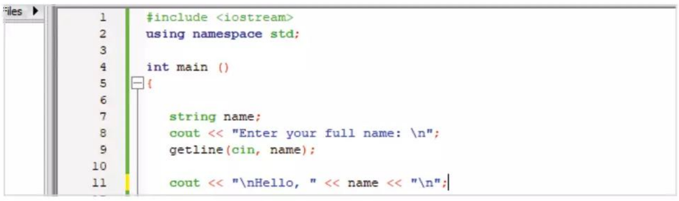


```
C:\Users\david\Documents\C++\userinteraction.exe
Enter a sentence:
BDM Publications Python and C++ for Beginners
Your sentence is: 45 characters long.

Process returned 0 (0x0)   execution time : 27.054 s
Press any key to continue.
```

STEP 10

Getline is usually a command that new C++ programmers forget to include. The terminating white space is annoying when you can't figure out why your code isn't working. In short, it's best to use getline(cin, variable) in future:

```
#include <iostream>
using namespace std;

int main ()
{

   string name;
   cout << “Enter your full name: \n”;
   getline(cin, name);

   cout << “\nHello, “ << name << “\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()
{

   string name;
   cout << "Enter your full name: \n";
   getline(cin, name);

   cout << "\nHello, " << name << "\n";
```

Character Literals

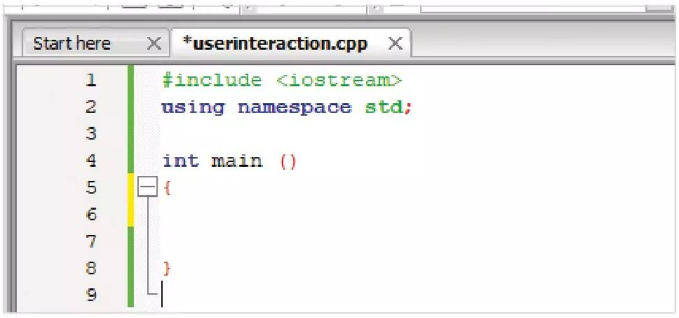
In C++ a literal is an object or variable that once defined remains the same throughout the code. However, a character literal is defined by a backslash, such as the \n you've been using at the end of a cout statement to signify a new line.

ESCAPE SEQUENCE

When used in something like a cout statement, character literals are also called Escape Sequence Codes. They allow you to insert a quote, an alert, new line and much more.

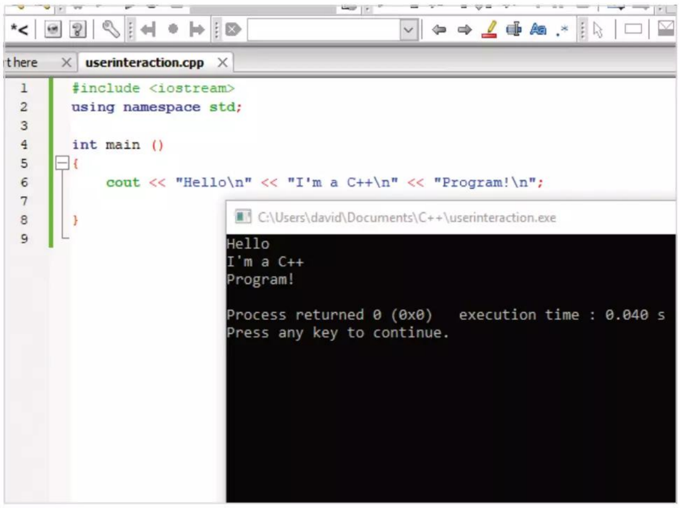
STEP 1 Create a new C++ file and enter the relevant headers:

```
#include <iostream>
using namespace std;

int main ()
{

}
```



```
#include <iostream>
using namespace std;

int main ()
{


}
```

STEP 2 You've already experienced the \n character literal placing a new line wherever it's called. The line: **cout << "Hello\n" << "I'm a C++\n" << "Program\n";** outputs three lines of text, each starting after the last \n.



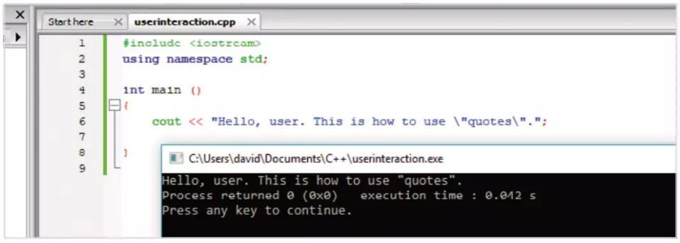
```
#include <iostream>
using namespace std;

int main ()
{
    cout << "Hello\n" << "I'm a C++\n" << "Program!\n";

}
```

C:\Users\david\Documents\C++\userinteraction.exe

```
Hello
I'm a C++
Program!

Process returned 0 (0x0)   execution time : 0.040 s
Press any key to continue.
```

STEP 3 If you wanted to insert speech quotes inside a cout statement, you would have to use a backslash as it already uses quotes:

```
#include <iostream>
using namespace std;

int main ()
{
    cout << "Hello, user. This is how to use 
\"quotes\".";

}
```



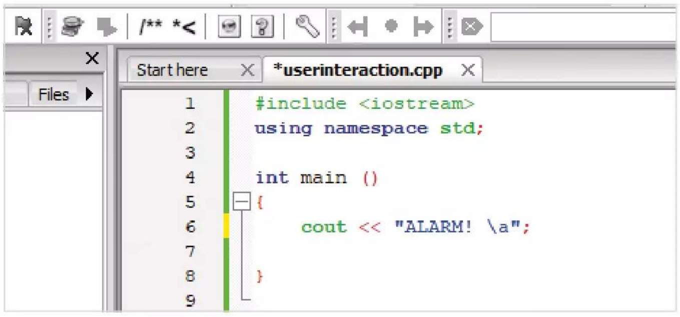
```
#include <iostream>
using namespace std;

int main ()
{
    cout << "Hello, user. This is how to use \"quotes\".";

}
```

C:\Users\david\Documents\C++\userinteraction.exe

```
Hello, user. This is how to use "quotes".
Process returned 0 (0x0)   execution time : 0.012 s
Press any key to continue.
```

STEP 4 There's even a character literal that can trigger an alarm. In Windows 10, it's the notification sound that chimes when you use \a. Try this code, and turn up your sound.

```
#include <iostream>
using namespace std;

int main ()
{
    cout << "ALARM!  \a";

}
```



```
#include <iostream>
using namespace std;

int main ()
{
    cout << "ALARM!  \a";

}
```

A HANDY CHART

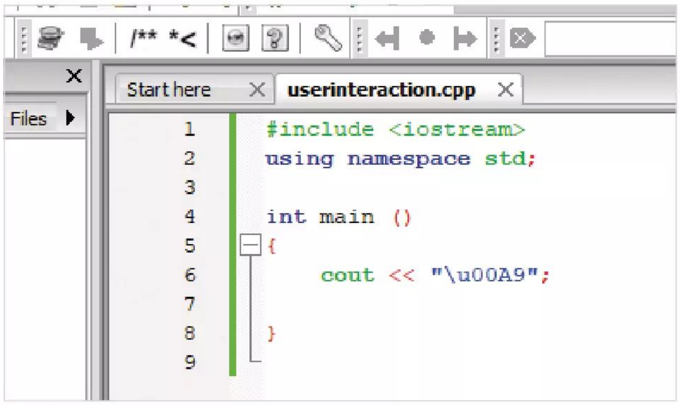
There are numerous character literals, or escape sequence codes, to choose from. We therefore thought it would be good for you to have a handy chart available, for those times when you need to insert a code.

ESCAPE SEQUENCE CODE	CHARACTER
\\	Backslash
\’	Single Quote
\”	Double Quote (Speech Marks)
\?	Question Mark
\a	Alert/Alarm
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Horizontal Tab
\v	Vertical Tab
\0	Null Character
\uxxxx	Unicode (UTF-8)
\Uxxxxxxxx	Unicode (UTF-16)

UNICODE CHARACTERS (UTF-8)

Unicode characters are symbols or characters that are standard across all platforms. For example, the copyright symbol, that can be entered via the keyboard by entering the Unicode code, followed by ALT+X. In the case of the copyright symbol enter: 00A9 Alt+X. In C++ code, you would enter:

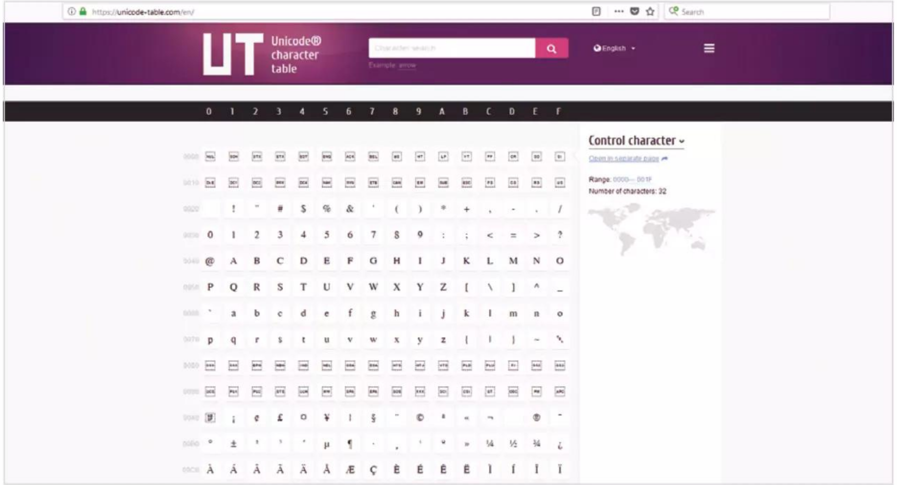
```
#include <iostream>
using namespace std;

int main ()
{
    cout << “\u00A9”;
```


Files
Start here
userinteraction.cpp

```
#include <iostream>
using namespace std;

int main ()
{
    cout << "\u00A9";

}
```

UNICODE CHARACTER TABLE

A complete list of the available Unicode characters can be found at **www. unicode-table.com/en/**. Hover your mouse over the character to see its unique code to enter in C++.


https://unicode-table.com/en/
Search
UT
Unicode®
character
table
English
Control character

Defining Constants

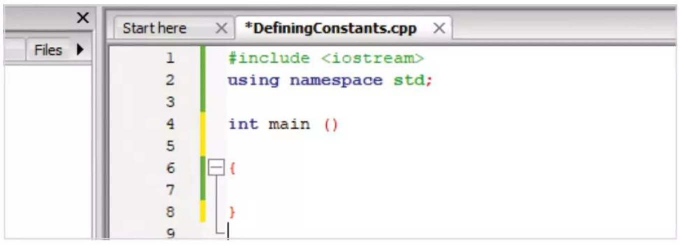
Constants are fixed values in your code. They can be any basic data type but as the name suggests their value remains constant throughout the entire code. There are two separate ways to define a constant in C++, the #define pre-processor and const.

#DEFINE

The pre-processors are instructions to the compiler to pre-process the information before it goes ahead and compiles the code. #include is a pre-processor as is #define.

STEP 1 You can use the #define pre-processor to define any constants you want in our code. Start by creating a new C++ file complete with the usual headers:

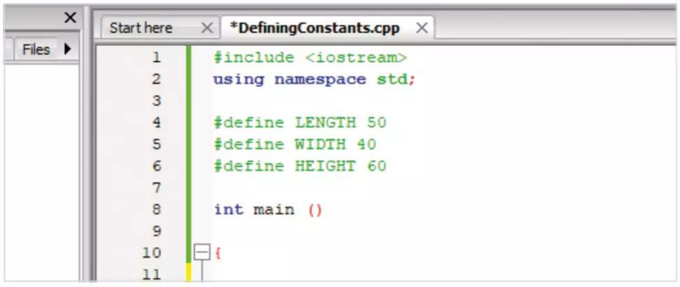
```
#include <iostream>
using namespace std;

int main ()

{

}
```



```
#include <iostream>
using namespace std;

int main ()

{

}
```

STEP 2 Now let's assume your code has three different constants: length, width and height. You can define them with:

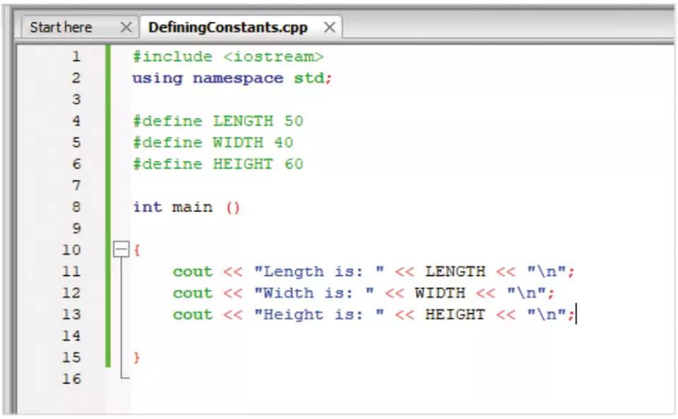
```
#include <iostream>
using namespace std;
#define LENGTH 50
#define WIDTH 40
#define HEIGHT 60

int main ()

{

}
```



```
#include <iostream>
using namespace std;

#define LENGTH 50
#define WIDTH 40
#define HEIGHT 60

int main ()

{
```

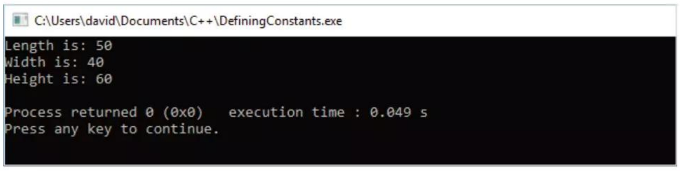
STEP 3 Note the capitals for defined constants, it's considered good programming practise to define all constants in capitals. Here, the assigned values are 50, 40 and 60, so let's call them up:

```
#include <iostream>
using namespace std;

#define LENGTH 50
#define WIDTH 40
#define HEIGHT 60

int main ()

{
    cout << “Length is: “ << LENGTH << “\n”;
    cout << “Width is: “ << WIDTH << “\n”;
    cout << “Height is: “ << HEIGHT << “\n”;
}
```



```
#include <iostream>
using namespace std;

#define LENGTH 50
#define WIDTH 40
#define HEIGHT 60

int main ()

{
    cout << "Length is: " << LENGTH << "\n";
    cout << "Width is: " << WIDTH << "\n";
    cout << "Height is: " << HEIGHT << "\n";

}
```

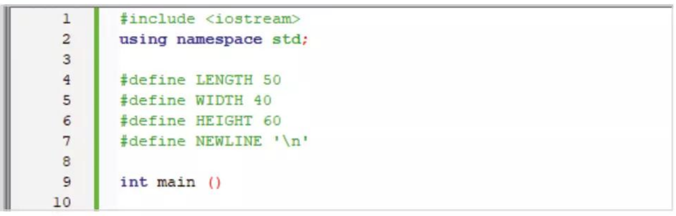
STEP 4 Build and run the code. Just as expected, it displays the values for each of the constants created. It's worth noting that you don't need a semicolon when you're defining a constant with the #define keyword.



```
Length is: 50
Width is: 40
Height is: 60

Process returned 0 (0x0)   execution time : 0.049 s
Press any key to continue.
```

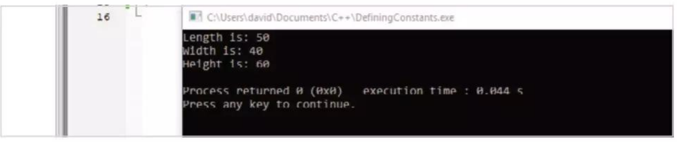
STEP 5 You can also define other elements as a constant. For example, instead of using \n for a newline in the cout statement, you can define it at the start of the code:

```
#include <iostream>
using namespace std;

#define LENGTH 50
#define WIDTH 40
#define HEIGHT 60
#define NEWLINE ‘\n’

int main ()

{
    cout << “Length is: “ << LENGTH << NEWLINE;
    cout << “Width is: “ << WIDTH << NEWLINE;
    cout << “Height is: “ << HEIGHT << NEWLINE;
}
```



```
#include <iostream>
using namespace std;

#define LENGTH 50
#define WIDTH 40
#define HEIGHT 60
#define NEWLINE '\n'

int main ()
```

STEP 6 The code, when built and executed, does exactly the same as before, using the new constant NEWLINE to insert a newline in the cout statement. Incidentally, creating a newline constant isn’t a good idea unless you’re making it smaller than \n or even the endl command.

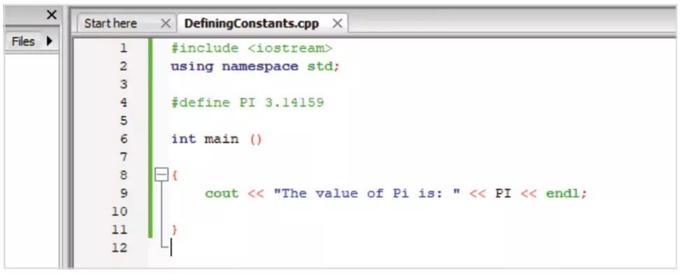


C:\Users\david\Documents\C++\DefiningConstants.exe

```
Length is: 50
Width is: 40
Height is: 60

Process returned 0 (0x0)   execution time : 0.044 s
Press any key to continue.
```

STEP 7 Defining a constant is a good way of initialising your base values at the start of your code. You can define that your game has three lives, or even the value of PI without having to call up the C++ math library:

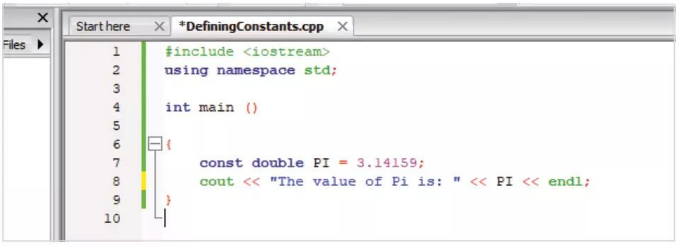
```
#include <iostream>
using namespace std;

#define PI 3.14159

int main ()

{
    cout << “The value of Pi is: “ << PI << endl;

}
```



```
#include <iostream>
using namespace std;

#define PI 3.14159

int main ()

{

    cout << "The value of Pi is: " << PI << endl;

}
```

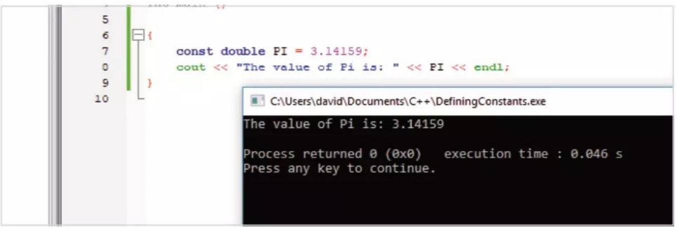
STEP 8 Another method of defining a constant is with the const keyword. Use const together with a data type, variable and value: const type variable = value. Using Pi as an example:

```
#include <iostream>
using namespace std;

int main ()

{
    const double PI = 3.14159;
    cout << “The value of Pi is: “ << PI << endl;
}
```



```
#include <iostream>
using namespace std;

int main ()

{

    const double PI = 3.14159;
    cout << "The value of Pi is: " << PI << endl;
}
```

STEP 9 Because you’re using const within the main block of code, you need to finish the line with a semicolon. You can use either, as long as the names and values don’t clash, but it’s worth mentioning that #define requires no memory, so if you’re coding to a set amount of memory, #define is your best bet.



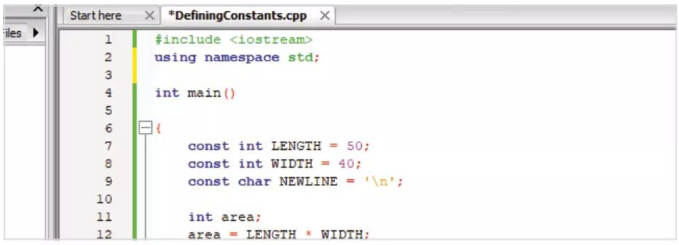
```

{

    const double PI = 3.14159;
    cout << "The value of Pi is: " << PI << endl;
}
```

C:\Users\david\Documents\C++\DefiningConstants.exe

```
The value of Pi is: 3.14159

Process returned 0 (0x0)   execution time : 0.046 s
Press any key to continue.
```

STEP 10 Const works in much the same way as #define. You can create static integers and even newlines:

```
#include <iostream>
using namespace std;

int main()

{
    const int LENGTH = 50;
    const int WIDTH = 40;
    const char NEWLINE = ‘\n’;

    int area;
    area = LENGTH * WIDTH;

    cout << “Area is: “ << area << NEWLINE;

}
```



```
#include <iostream>
using namespace std;

int main()

{
    const int LENGTH = 50;
    const int WIDTH = 40;
    const char NEWLINE = '\n';

    int area;
    area = LENGTH * WIDTH;
```

File Input/Output

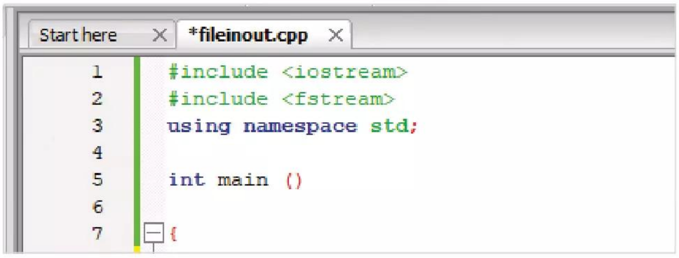
The standard iostream library provides C++ coders with the cin and cout input and output functionality. However, to be able to read and write from a file you need to utilise another C++ library, called fstream.

FSTREAMS

There are two main data types within the fstream library that are used to open a file, read from it and write to it, ofstream and ifstream. Here's how they work.

STEP 1 The first task is to create a new C++ file and along with the usual headers you need to include the new fstream header:

```
#include <iostream>
#include <fstream>
Using namespace std;

int main ()

{



}
```

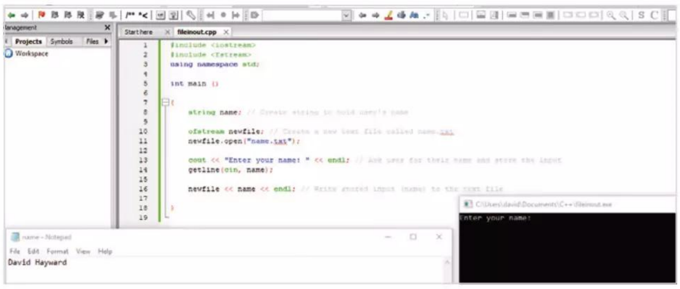


```
#include <iostream>
#include <fstream>
using namespace std;

int main ()

{
```

STEP 2 Begin by asking a user for their name and writing that information to a file. You need the usual string to store the name, and getline to accept the input from the user.

```
#include <iostream>
#include <fstream>
using namespace std;

int main ()

{

    string name;

    ofstream newfile;
    newfile.open("name.txt");

    cout << "Enter your name: " << endl;
    getline(cin, name);

    newfile << name << endl;

    newfile.close();

}
```

STEP 3 We've included comments in the screenshot of step 2 to help you understand the process. You created a string called name, to store the user's inputted name. You also created a text file called name.txt (with the ofstream newfile and newfile.open lines), asked the user for their name and stored it and then written the data to the file.



Start here
fileinout.cpp
Projects
Symbols
Files
Workspace
name - Notepad
File Edit Format View Help
C:\Users\david\Documents\C++\fileinout.exe

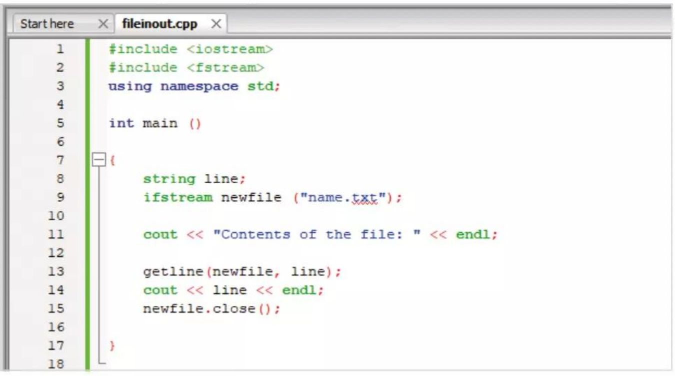
```
#include <iostream>
#include <fstream>
using namespace std;

int main ()

{

    string name; // Create string to hold user's name

    ofstream newfile; // Create a new text file called name.txt
    newfile.open("name.txt");

    cout << "Enter your name: " << endl; // Ask user for their name and store the input
    getline(cin, name);

    newfile << name << endl; // Write stored input (name) to the text file

}
```

```
David Hayward
```

```
Enter your name:
```

STEP 4 To read the contents of a file, and output it to the screen, you need to do things slightly differently. First you need to create a string variable to store the file's contents (line by line), then open the file, use getline to read the file line by line and output those lines to the screen. Finally, close the file.

```
    string line;
    ifstream newfile ("name.txt");

    cout << "Contents of the file: " << endl;

    getline(newfile, line);
    cout << line << endl;
    newfile.close();
```

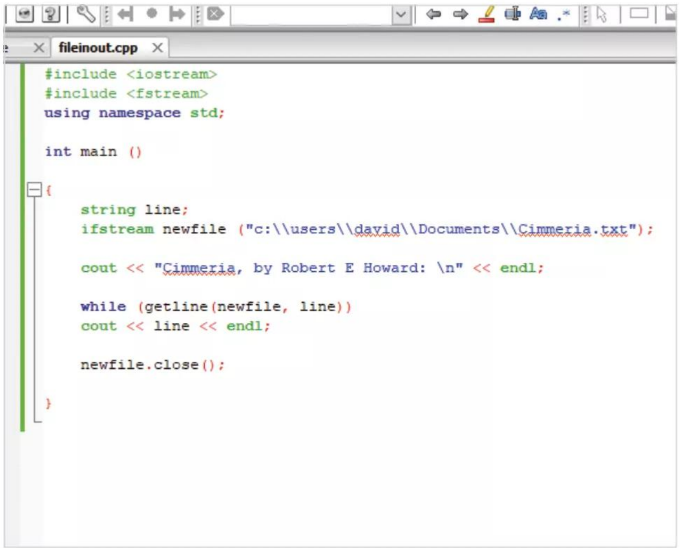


```
#include <iostream>
#include <fstream>
using namespace std;

int main ()

{
    string line;
    ifstream newfile ("name.txt");

    cout << "Contents of the file: " << endl;

    getline(newfile, line);
    cout << line << endl;
    newfile.close();

}
```

STEP 5

The code above is great for opening a file with one or two lines but what if there are multiple lines? Here we opened a text file of the poem Cimmeria, by Robert E Howard:

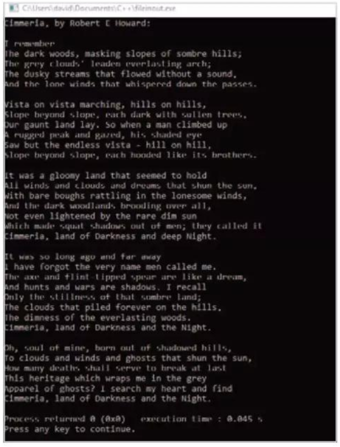
```
    string line;
    ifstream newfile (“c:\\users\\david\\
Documents\\Cimmeria.txt”);

    cout << “Cimmeria, by Robert E Howard: \n” <<
endl;

    while (getline(newfile, line))
    cout << line << endl;

    newfile.close();
```



```
#include <iostream>
#include <fstream>
using namespace std;

int main ()

{

    string line;
    ifstream newfile ("c:\\users\\david\\Documents\\Cimmeria.txt");

    cout << "Cimmeria, by Robert E Howard: \n" << endl;

    while (getline(newfile, line))
    cout << line << endl;

    newfile.close();

}
```

STEP 6

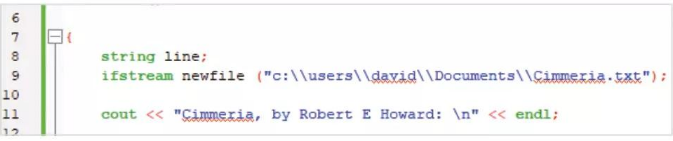
You can no doubt see that we've included a while loop, which we cover in a few pages time. It means that while there are lines to be read from the text file, C++ getlines them. Once all the lines are read, the output is displayed on the screen and the file is closed.



```
Cimmeria, by Robert E Howard:

I remember
The dark woods, masking slopes of sombre hills;
The grey clouds' leaden everlasting arch;
The dusky streams that flowed without a sound,
And the lone winds that whispered down the passes.

Vista on vista marching, hills on hills,
Slope beyond slope, each dark with sullen trees,
Our gaunt land lay. So when a man climbed up
A rugged peak and gazed, his shaded eye
Saw but the endless vista - hill on hill,
Slope beyond slope, each hooded like its brothers.

It was a gloomy land that seemed to hold
All winds and clouds and dreams that shun the sun,
With bare boughs rattling in the lonesome winds,
And the dark woodlands brooding over all,
Not even lightened by the rare dim sun
Which made squat shadows out of men; they called it
Cimmeria, land of Darkness and deep Night.

It was so long ago and far away
I have forgot the very name men called me.
The axe and flint-tipped spear are like a dream,
And hunts and wars are shadows. I recall
Only the stillness of that sombre land;
The clouds that piled forever on the hills,
The dimness of the everlasting woods.
Cimmeria, land of Darkness and the Night.

Oh, soul of mine, born out of shadowed hills,
To clouds and winds and ghosts that shun the sun,
How many deaths shall serve to break at last
This heritage which wraps me in the grey
Apparel of ghosts? I search my heart and find
Cimmeria, land of Darkness and the Night.

Process returned 0 (0x0)   execution time : 0.045 s
Press any key to continue.
```

STEP 7

You can also see that the location of the text file Cimmeria.txt isn't in the same folder as the C++ program. When we created the first name.txt file, it was written to the same folder where the code was located; this is done by default. To specify another folder, you need to use double-back slashes, as per the character literals/escape sequence code.

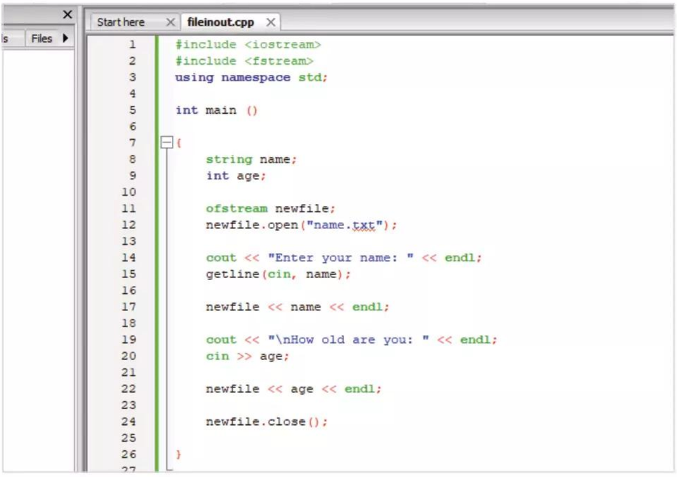


```
{

    string line;
    ifstream newfile ("c:\\users\\david\\Documents\\Cimmeria.txt");

    cout << "Cimmeria, by Robert E Howard: \n" << endl;
```

STEP 8

Just as you might expect, you can write almost anything you like to a file, for reading either in Notepad or via the console through the C++ code:

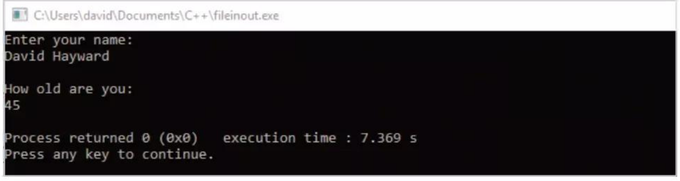
```
    string name;
    int age;

    ofstream newfile;
    newfile.open(“name.txt”);

    cout << “Enter your name: “ << endl;
    getline(cin, name);

    newfile << name << endl;

    cout << “\nHow old are you: “ << endl;
    cin >> age;

    newfile << age << endl;

    newfile.close();
```



```
#include <iostream>
#include <fstream>
using namespace std;

int main ()

{

    string name;
    int age;

    ofstream newfile;
    newfile.open("name.txt");

    cout << "Enter your name: " << endl;
    getline(cin, name);

    newfile << name << endl;

    cout << "\nHow old are you: " << endl;
    cin >> age;

    newfile << age << endl;

    newfile.close();

}
```

STEP 9

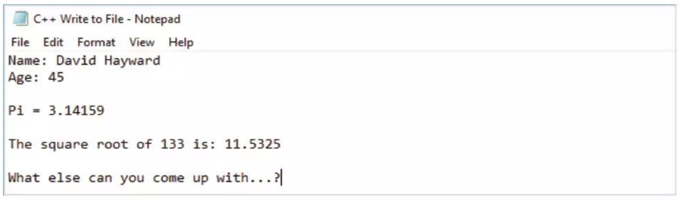
The code from step 8 differs again, but only where it comes to adding the age integer. Notice that we used cin >> age, instead of the previous getline(cin, variable). The reason for this is that the getline function handles strings, not integers; so when you're using a data type other than a string, use the standard cin.



```
Enter your name:
David Hayward

How old are you:
45

Process returned 0 (0x0)   execution time : 7.369 s
Press any key to continue.
```

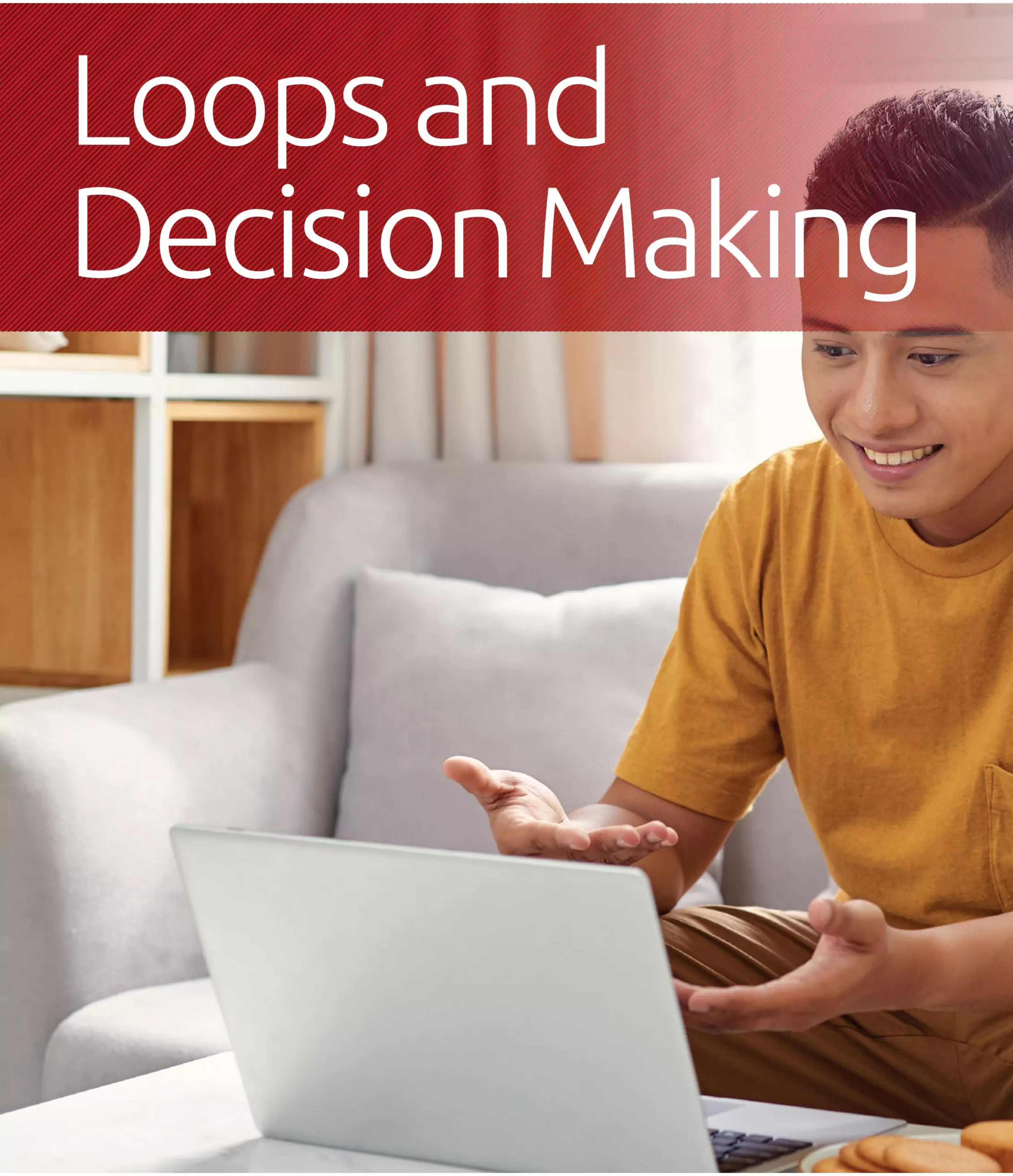
STEP 10

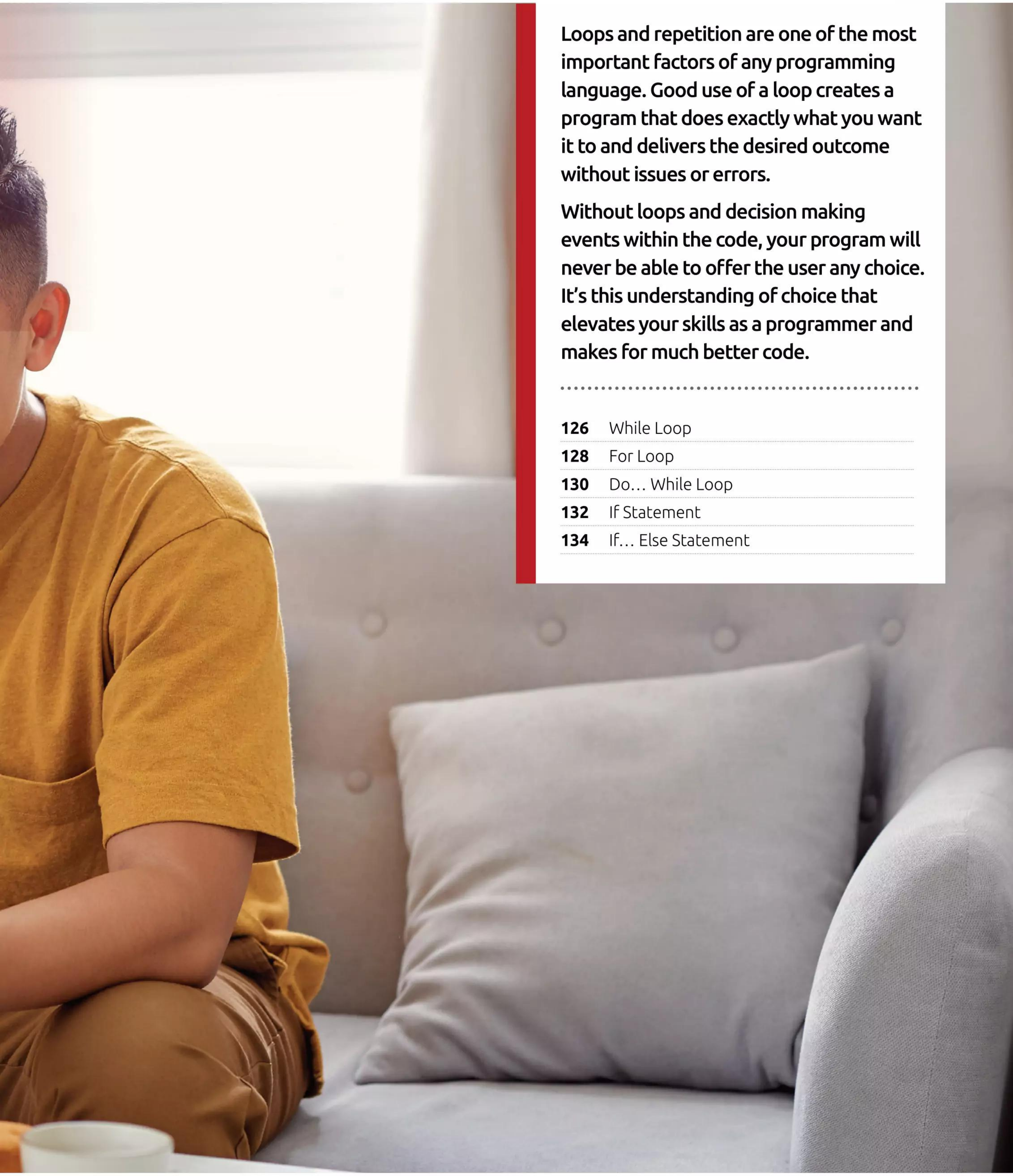
Here's an exercise: see if you can create code to write several different elements to a text file. You can have a user's name, age, phone number etc. Maybe even the value of Pi and various mathematical elements. It's all good practice.



```
Name: David Hayward
Age: 45

Pi = 3.14159

The square root of 133 is: 11.5325

What else can you come up with...?
```

Loops and Decision Making





Loops and repetition are one of the most important factors of any programming language. Good use of a loop creates a program that does exactly what you want it to and delivers the desired outcome without issues or errors.

Without loops and decision making events within the code, your program will never be able to offer the user any choice. It's this understanding of choice that elevates your skills as a programmer and makes for much better code.

126 While Loop
128 For Loop
130 Do… While Loop
132 If Statement
134 If… Else Statement

While Loop

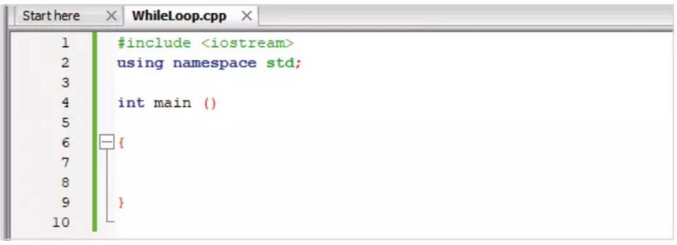
A while loop's function is to repeat a statement, or a group of statements, while a certain condition remains true. When the while loop starts, it initialises itself by testing the condition of the loop and the statements within, before executing the rest of the loop.

TRUE OR FALSE?

While loops are one of the most popular form of C++ code looping. They repeatedly run the code contained within the loop while the condition is true. Once it proves false, the code continues as normal.

STEP 1 Clear what you've done so far and create a new C++ file. There's no need for any extra headers at the moment, so add the standard headers as per usual:

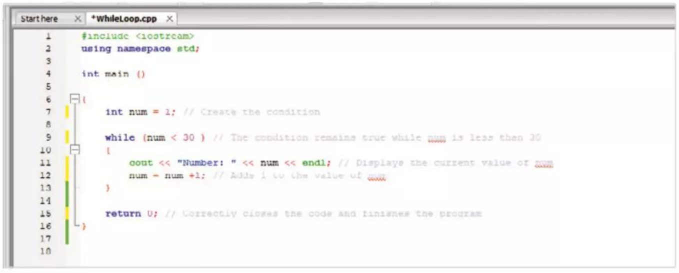
```
#include <iostream>
using namespace std;

int main ()

{

}
```



```
#include <iostream>
using namespace std;

int main ()

{


}

```

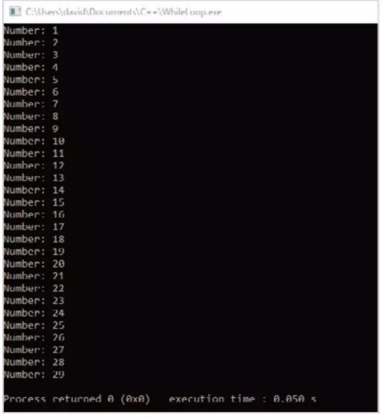
STEP 2 Create a simple while loop. Enter the code below, build and run (we've added comments to the screen shot):

```
{
    int num = 1;

    while (num < 30 )
    {
        cout << “Number: “ << num << endl;
        num = num +1;
    }

    return 0;
}
```



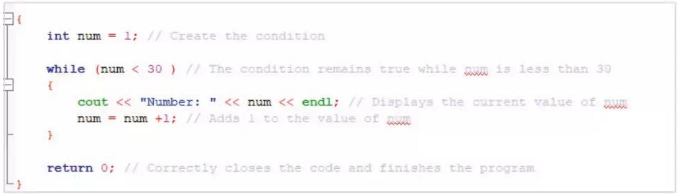
```
#include <iostream>
using namespace std;

int main ()

{
    int num = 1; // Create the condition

    while (num < 30 ) // The condition remains true while num is less than 30
    {
        cout << "Number: " << num << endl; // Displays the current value of num
        num = num +1; // Adds 1 to the value of num
    }

    return 0; // Correctly closes the code and finishes the program
}

```

STEP 3 First you need to create a condition, so use a variable called num and give it the value 1. Now create the while loop, stating that as long as num is less than 30, the loop is true. Within the loop the value of num is displayed and adds 1 until it's more than 30.



```
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
Number: 6
Number: 7
Number: 8
Number: 9
Number: 10
Number: 11
Number: 12
Number: 13
Number: 14
Number: 15
Number: 16
Number: 17
Number: 18
Number: 19
Number: 20
Number: 21
Number: 22
Number: 23
Number: 24
Number: 25
Number: 26
Number: 27
Number: 28
Number: 29

Process returned 0 (0x0)   execution time : 0.050 s
```

STEP 4 We're introducing a few new elements here. The first are the opening and closing braces for the while loop. This is because our loop is a compound statement, meaning a group of statements; note also, there's no semicolon after the while statement. You now also have return 0, which is a clean and preferred way of ending the code.



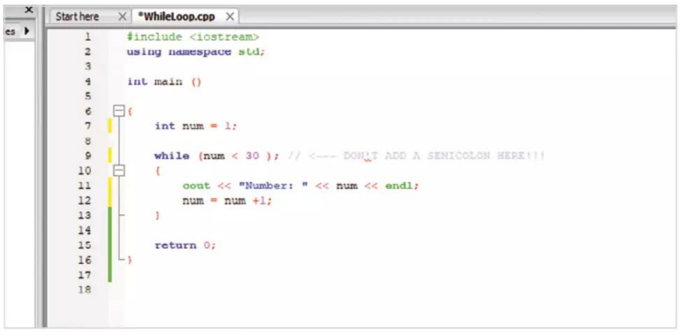
```
{
    int num = 1; // Create the condition

    while (num < 30 ) // The condition remains true while num is less than 30
    {
        cout << "Number: " << num << endl; // Displays the current value of num
        num = num +1; // Adds 1 to the value of num
    }

    return 0; // Correctly closes the code and finishes the program
}
```

STEP 5 If you didn't need to see the continually increasing value of num, you could have done away with the compound while statement and instead just added num by itself until it reached 30, and then displayed the value:

```
{
    int num = 1;

    while (num < 30 )
        num++;
    cout << “Number: “ << num << endl;

    return 0;
}
```

STEP 6

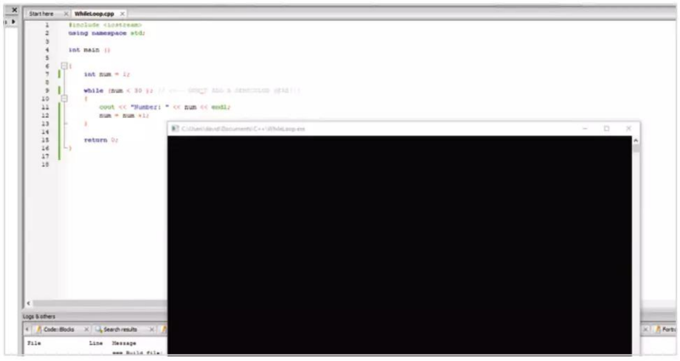
It's important to remember not to add a semicolon at the end of a while statement. Why? Well, as you know, the semicolon represents the end of a C++ line of code. If you place one at the end of a while statement, your loop will be permanently stuck until you close the program.



```
#include <iostream>
using namespace std;

int main ()

{

    int num = 1;

    while (num < 30 ); // <--- DON'T ADD A SEMICOLON HERE!!!
    {
        cout << "Number: " << num << endl;
        num = num +1;
    }

    return 0;
}
```

STEP 7

In our example, if we were to execute the code the value of num would be 1, as set by the int statement. When the code hits the while statement it reads that while the condition of 1 being less than 30 is true, loop. The semicolon closes the line, so the loop repeats; but it never adds 1 to num, as it won't continue through the compound statement.



STEP 8

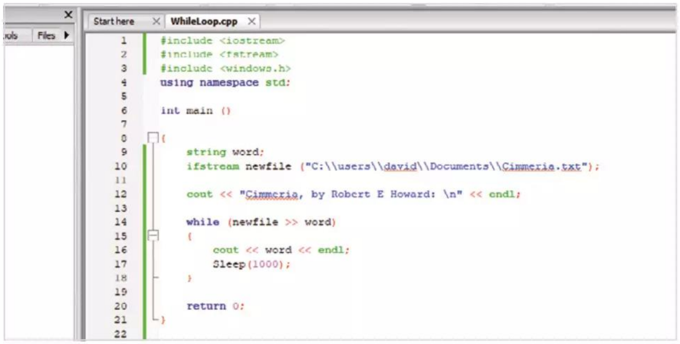
You can manipulate the while statement to display different results depending on what code lies within the loop. For example, to read the poem, Cimmeria, word by word, you would enter:

```
#include <iostream>
#include <fstream>
using namespace std;

int main ()

{
    string word;
    ifstream newfile (“C:\\users\\david\\
Documents\\Cimmeria.txt”);

    cout << “Cimmeria, by Robert E Howard: \n” <<
endl;

    while (newfile >> word)
    {
        cout << word << endl;
    }

    return 0;
}
```

STEP 9

You can further expand the code to enable each word of the poem to appear every second. To do so, you need to pull in a new library, <windows.h>. This is a Windows only library and within it you can use the Sleep() function:

```
#include <iostream>
#include <fstream>
#include <windows.h>
using namespace std;

int main ()

{
    string word;
    ifstream newfile (“C:\\users\\david\\
Documents\\Cimmeria.txt”);

    cout << “Cimmeria, by Robert E Howard: \n” <<
endl;

    while (newfile >> word)
    {
        cout << word << endl;
        Sleep(1000);
    }

    return 0;
}
```



```
#include <iostream>
#include <fstream>
#include <windows.h>
using namespace std:

int main ()

{
    string word;
    ifstream newfile ("C:\\users\\david\\Documents\\Cimmeria.txt");

    cout << "Cimmeria, by Robert E Howard: \n" << endl;

    while (newfile >> word)
    {
        cout << word << endl;
        Sleep(1000);
    }

    return 0:
}
```

STEP 10

Sleep() works in milliseconds, so Sleep(1000) is one second, Sleep(10000) is ten seconds and so on. Combining the sleep function (along with the header it needs) and a while loop enables you to come up with some interesting countdown code.

```
#include <iostream>
#include <windows.h>
using namespace std;

int main ()

{
    int a = 10;

    while (a != 0)
    {
        cout << a << endl;
        a = a - 1;
        Sleep(1000);
    }

    cout << “\nBlast Off!” << endl;

    return 0;
}
```

For Loop

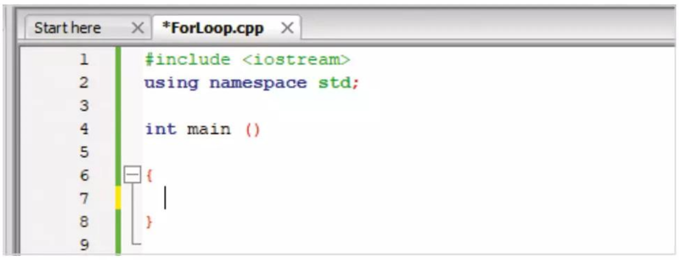
In some respects, a for loop works in a very similar way to that of a while loop, although it's structure is different. A for loop is split into three stages: an initialiser, a condition and an incremental step. Once set up, the loop repeats itself until the condition becomes false.

LOOPY LOOPS

The initialise stage of a for loop is executed only once and this sets the point reference for the loop. The condition is evaluated by the loop to see if it's true or false and then the increment is executed. The loop then repeats the second and third stage.

STEP 1 Create a new C++ file, with the standard headers:

```
#include <iostream>
using namespace std;

int main ()

{


}
```

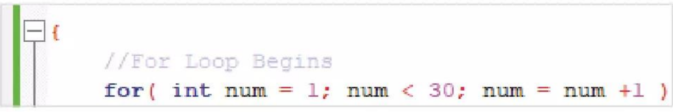


```
#include <iostream>
using namespace std;

int main ()

{

}
```

STEP 2 Start simple and create a for loop that counts from 1 to 30, displaying the value to the screen with each increment:

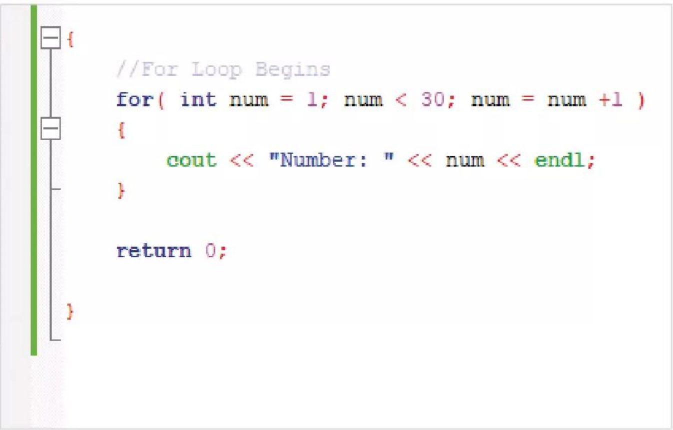
```
{
    //For Loop Begins
    for( int num = 1; num < 30; num = num +1 )
    {
        cout << “Number: “ << num << endl;
    }

    return 0;

}
```

STEP 3 Working through the process of the for loop, begin by creating an integer called num and assigning it a value of 1. Next, set the condition, in this case num being less than 30. The last stage is where you create the increments; here it's the value of num being added by 1.



```
{

    //For Loop Begins
    for( int num = 1; num < 30; num = num +1 )
```

STEP 4 After the loop, you created a compound statement in braces (curly brackets), that displays the current value of the integer num. Every time the for loop repeats itself, the second and third stages of the loop, it adds 1 until the condition <30 is false. The loop then ends and the code continues, ending neatly with return 0.

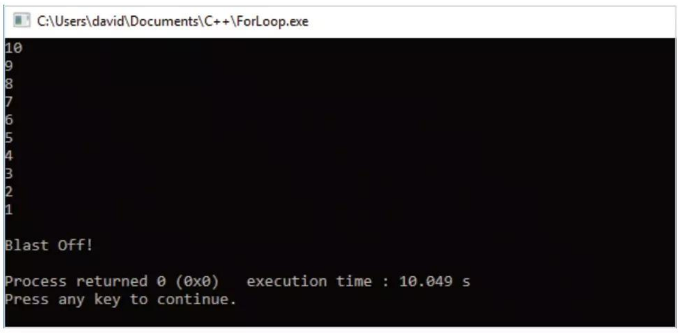


```
{

    //For Loop Begins
    for( int num = 1; num < 30; num = num +1 )
    {
        cout << "Number: " << num << endl;
    }

    return 0;


}
```

STEP 5 A for loop is quite a neat package in C++, all contained within its own brackets, while the other elements outside of the loop are displayed below. If you want to create a 10-second countdown, you could use:

```
#include <iostream>
#include <windows.h>
using namespace std;

int main ()

{
    //For Loop Begins
    for( int a = 10; a != 0; a = a -1 )
    {
        cout << a << endl;
        Sleep(1000);
    }

    cout << “\nBlast Off!” << endl;

    return 0;

}
```

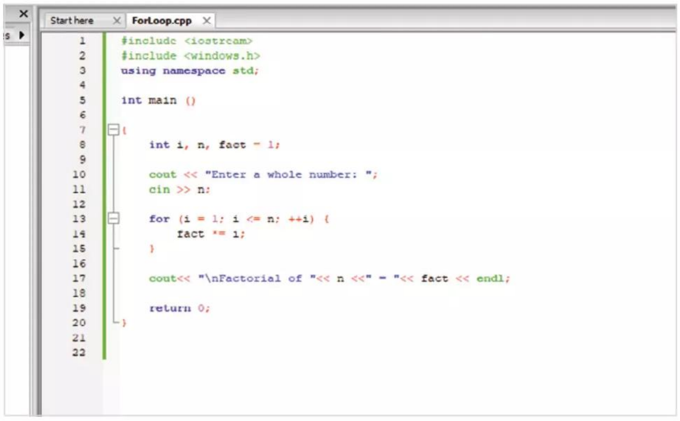
STEP 6 With the countdown code, don't forget to include the windows.h library, so you can use the Sleep command. Build and run the code; in the command console you can see the numbers 10 to 1 countdown in one second increments, until it reaches zero and Blast Off! appears.



```
C:\Users\david\Documents\C++\ForLoop.exe
10
9
8
7
6
5
4
3
2
1

Blast Off!

Process returned 0 (0x0)   execution time : 10.049 s
Press any key to continue.
```

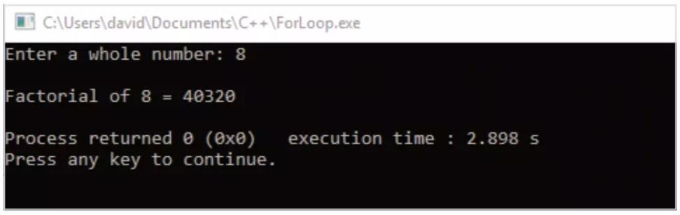
STEP 7 Naturally you can include a lot more content into a for loop, including some user input:

```
    int i, n, fact = 1;

    cout << “Enter a whole number: “;
    cin >> n;

    for (i = 1; i <= n; ++i) {
        fact *= i;
    }

    cout<< “\nFactorial of “<< n <<” = “<< fact <<
endl;

    return 0;
```



```
#include <iostream>
#include <windows.h>
using namespace std;

int main ()

{
    int i, n, fact = 1;

    cout << "Enter a whole number: ";
    cin >> n;

    for (i = 1; i <= n; ++i) {
        fact *= i;
    }

    cout<< "\nFactorial of "<< n <<" = "<< fact << endl;

    return 0;
}
```

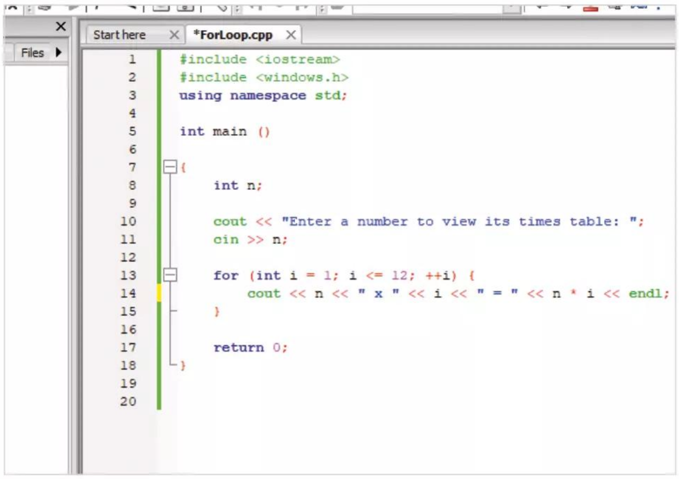
STEP 8 The code from step 7, when built and run, asks for a number, then displays the factorial of that number through the for loop. The user's number is stored in the integer n, followed by the integer I which is used to check if the condition is true or false, adding 1 each time and comparing it to the user's number, n.



```
C:\Users\david\Documents\C++\ForLoop.exe
Enter a whole number: 8

Factorial of 8 = 40320

Process returned 0 (0x0)   execution time : 2.898 s
Press any key to continue.
```

STEP 9 Here's an example of a for loop displaying the multiplication tables of a user inputted number. Handy for students:

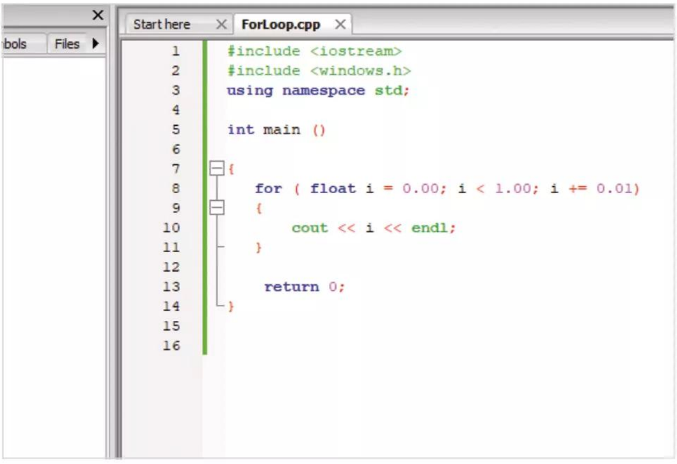
```
{
    int n;

    cout << “Enter a number to view its times
table: “;
    cin >> n;

    for (int i = 1; i <= 12; ++i) {
        cout << n << “ x “ << i << “ = “ << n * i
<< endl;
    }

    return 0;
}
```



```
#include <iostream>
#include <windows.h>
using namespace std;

int main ()

{
    int n;

    cout << "Enter a number to view its times table: ";
    cin >> n;

    for (int i = 1; i <= 12; ++i) {
        cout << n << " x " << i << " = " << n * i << endl;
    }

    return 0;
}
```

STEP 10 The value of the integer i can be expanded from 12 to whatever number you want, displaying a very large multiplication table in the process (or a small one). Of course the data type within a for loop doesn't have to be an integer; as long as it's valid, it works.

```
for ( float i = 0.00; i < 1.00; i += 0.01)
    {
        cout << i << endl;
    }
return 0;
```



```
#include <iostream>
#include <windows.h>
using namespace std;

int main ()

{
    for ( float i = 0.00; i < 1.00; i += 0.01)
    {
        cout << i << endl;
    }

    return 0;
}
```

Do… While Loop

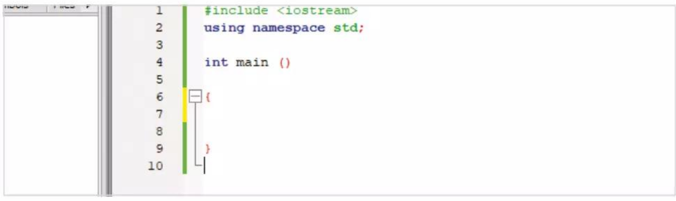
A do… while loop differs slightly from that of a for or even a while loop. Both for and while set and examine the state of the condition at the start of the loop, or the top of the loop if you prefer. However, a do… while loop, is similar to a while loop but instead checks the condition at the bottom of the loop.

DO LOOPS

The good thing about a do… while loop is that it's guaranteed to run through at least once. It's structure is: do, followed by statements, while condition is true. This is how it works.

STEP 1 Begin with a new, blank C++ file and enter the standard headers:

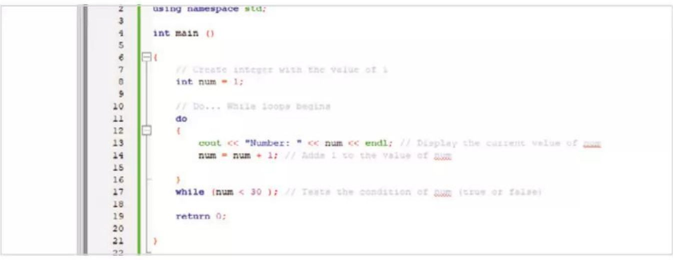
```
#include <iostream>
using namespace std;

int main ()

{


}
```



```
#include <iostream>
using namespace std;

int main ()

{


}
```

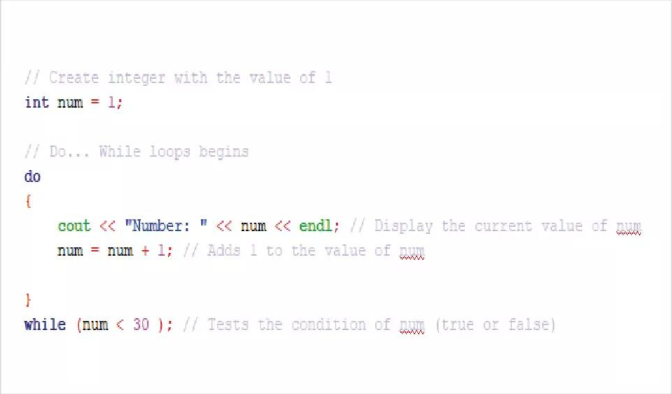
STEP 2 Begin with a simple number count:

```
{
    int num = 1;

    do
    {
        cout << “Number: “ << num << endl;
        num = num + 1;

    }
    while (num < 30 );

    return 0;

}
```



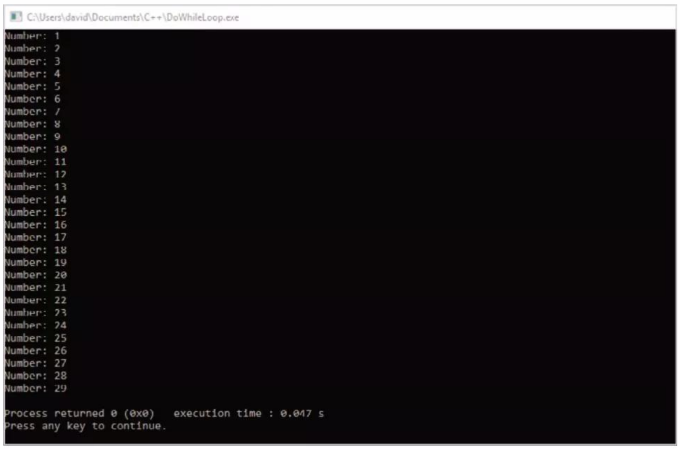
```
using namespace std;

int main ()

{
    // Create integer with the value of 1
    int num = 1;

    // Do... While loops begins
    do
    {
        cout << "Number: " << num << endl; // Display the current value of num
        num = num + 1; // Adds 1 to the value of num

    }
    while (num < 30 ); // Tests the condition of num (true or false)

    return 0;

}
```

STEP 3 Now, here's a look at the structure of a do… while loop. First you create an integer called num, with the value of 1. Now the do… while loops begins. The code inside the body of the loop is executed at least once, then the condition is checked for either true or false.



```
// Create integer with the value of 1
int num = 1;

// Do... While loops begins
do
{
    cout << "Number: " << num << endl; // Display the current value of num
    num = num + 1; // Adds 1 to the value of num

}
while (num < 30 ); // Tests the condition of num (true or false)
```

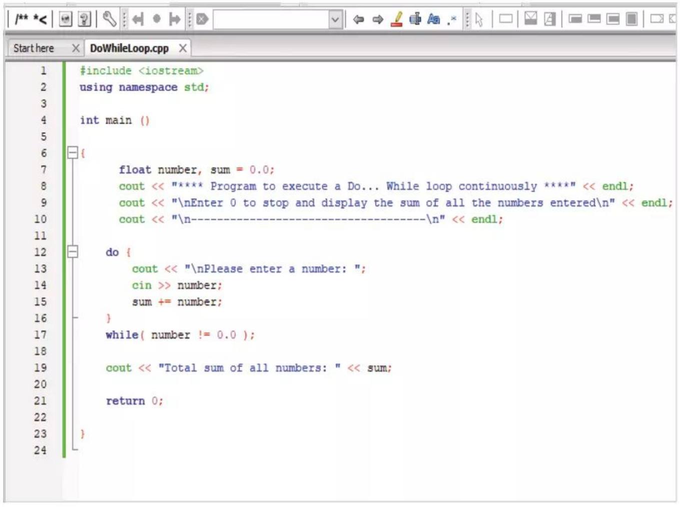
STEP 4 If the condition is true, the loop is executed. This continues until the condition is false. When the condition has been expressed as false, the loop terminates and the code continues. This means you can create a loop where the code continues until the user enters a certain character.



```
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
Number: 6
Number: 7
Number: 8
Number: 9
Number: 10
Number: 11
Number: 12
Number: 13
Number: 14
Number: 15
Number: 16
Number: 17
Number: 18
Number: 19
Number: 20
Number: 21
Number: 22
Number: 23
Number: 24
Number: 25
Number: 26
Number: 27
Number: 28
Number: 29

Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

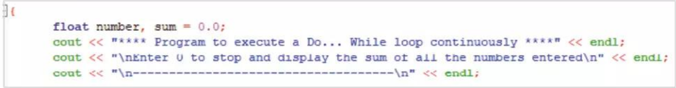
STEP 5 If you want code to add up user inputted numbers until the user enters zero:

```
{
      float number, sum = 0.0;
      cout << “**** Program to execute a Do...
While loop continuously ****” << endl;
      cout << “\nEnter 0 to stop and display the
sum of all the numbers entered\n” << endl;
      cout << “\n--------------------------------
---\n” << endl;

    do {
        cout<<”\nPlease enter a number: “;
        cin>>number;
        sum += number;
    }
    while(number != 0.0);

    cout<<”Total sum of all numbers: “<<sum;

    return 0;

}
```



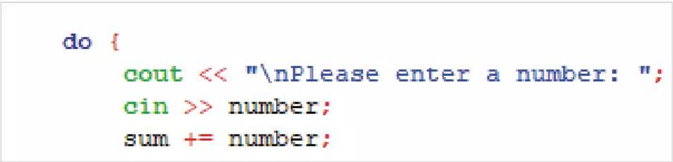
```
#include <iostream>
using namespace std;

int main ()

{

        float number, sum = 0.0;
        cout << "**** Program to execute a Do... While loop continuously ****" << endl;
        cout << "\nEnter 0 to stop and display the sum of all the numbers entered\n" << endl;
        cout << "\n-----------------------------------\n" << endl;

      do {
          cout << "\nPlease enter a number: ";
          cin >> number;
          sum += number;
      }
      while( number != 0.0 );

      cout << "Total sum of all numbers: " << sum;

      return 0;

}
```

STEP 6 The code from Step 5 works as follows: two floating point variables are assigned, number and sum, both with the value of 0.0. There is a brief set of instructions for the user, then the do… while loop begins.



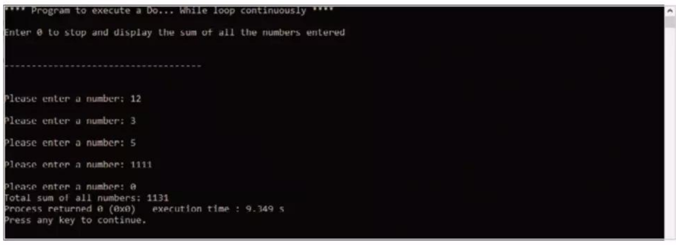
```
{
        float number, sum = 0.0;
        cout << "**** Program to execute a Do... While loop continuously ****" << endl;
        cout << "\nEnter 0 to stop and display the sum of all the numbers entered\n" << endl;
        cout << "\n-----------------------------------\n" << endl;
```

STEP 7 The do… while loop in this instance asks the user to input a number, which you assigned to the float variable, number. The calculation step uses the second floating point variable, sum, which adds the value of number every time the user enters a new value.



```
do {
    cout << "\nPlease enter a number: ";
    cin >> number;
    sum += number;
```

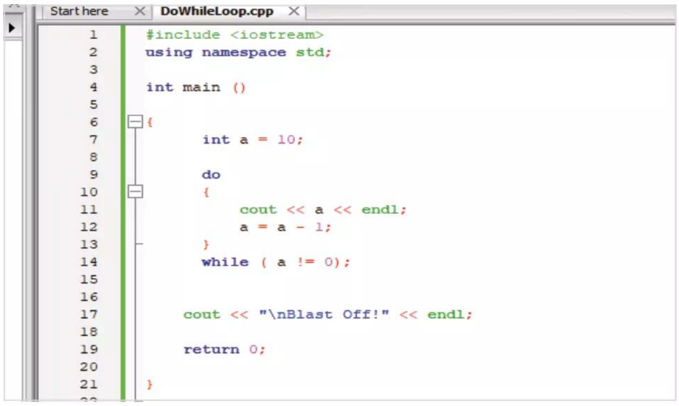
STEP 8 Finally, the while statement checks the condition of the variable number. If the user has entered zero, then the loop is terminated, if not then it continues indefinitely. When the user finally enters zero, the value of sum, the total value of all the user's input, is displayed. The loop, and the program, then ends.



```
**** Program to execute a Do... While loop continuously ****

Enter 0 to stop and display the sum of all the numbers entered


-----------------------------------


Please enter a number: 12

Please enter a number: 3

Please enter a number: 5

Please enter a number: 1111

Please enter a number: 0
Total sum of all numbers: 1131
Process returned 0 (0x0)   execution time : 9.349 s
Press any key to continue.
```

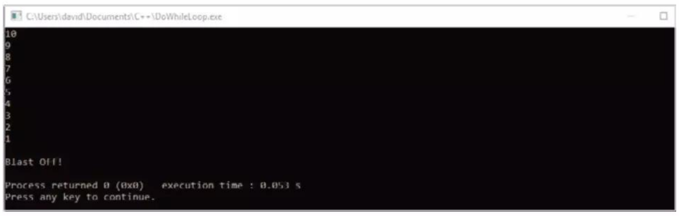
STEP 9 Using the countdown and Blast Off! code used previously, a do… while loop would look like:

```
{
      int a = 10;

      do
      {
          cout << a << endl;
          a = a - 1;
      }
      while ( a != 0);


    cout << “\nBlast Off!” << endl;

    return 0;

}
```



```
#include <iostream>
using namespace std;

int main ()

{

        int a = 10;

        do
        {
            cout << a << endl;
            a = a - 1;
        }
        while ( a != 0);


      cout << "\nBlast Off!" << endl;

      return 0;

}
```

STEP 10 The main advantage of using a do… while loop is because it's an exit-condition loop; whereas a while loop is an entry-control loop. Therefore, if your code requires a loop that needs to be executed at least once (for example, to check the number of lives in a game), then a do… while loop is perfect.



```
10
9
8
7
6
5
4
3
2
1

Blast Off!

Process returned 0 (0x0)   execution time : 0.053 s
Press any key to continue.
```

If Statement

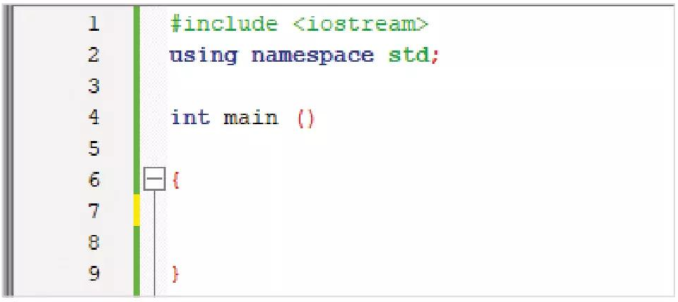
The decision making statement ‘if’ is probably one of the most used statements in any programming language, regardless of whether it’s C++, Python, BASIC or anything else. It represents a junction in the code, where IF one condition is true, do this; or IF it’s false, do that.

IF ONLY

If uses a Boolean expression within its statement. If the Boolean expression is true, the code within the statement is executed. If not, then the code after the statement is executed instead.

STEP 1 First, create a new C++ file and enter the relevant standard headers, as usual:

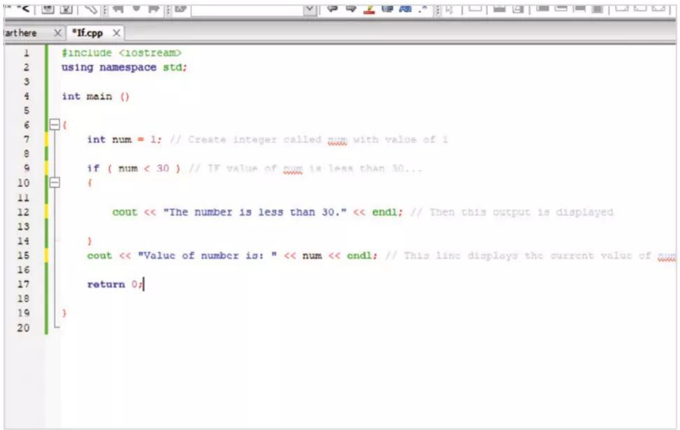
```
#include <iostream>
using namespace std;

int main ()

{



}
```



```
#include <iostream>
using namespace std;

int main  ()

{



}
```

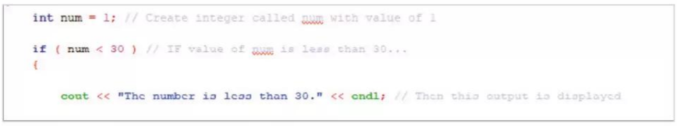
STEP 2 If is best explained when you use a number-based condition:

```
{
    int num = 1;

    if ( num < 30 )
    {

        cout << “The number is less than 30.” << endl;

    }
    cout << “Value of number is: “ << num << endl;

    return 0;

}
```



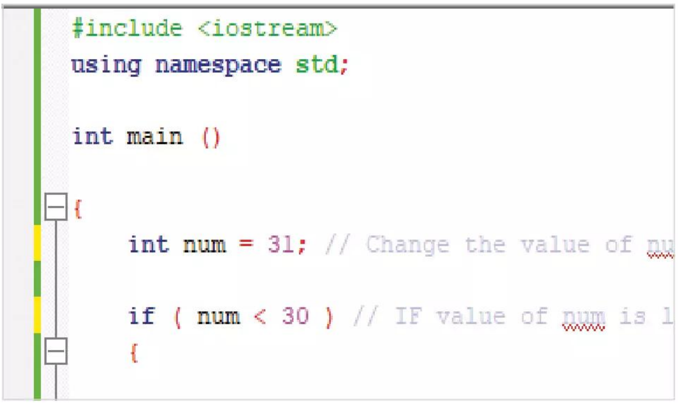
```
#include <iostream>
using namespace std;

int main ()

{

    int num = 1; // Create integer called num with value of 1

    if ( num < 30 ) // IF value of num is less than 30...
    {

        cout << "The number is less than 30." << endl; // Then this output is displayed

    }
    cout << "Value of number is: " << num << endl; // This line displays the current value of num

    return 0;

}
```

STEP 3 What’s going on here? To begin, an integer called num was created and assigned with the value of 1. The if statement comes next, and in this case we’ve instructed the code that if the condition, the value, of num is less than 1, then the code within the braces should be executed.



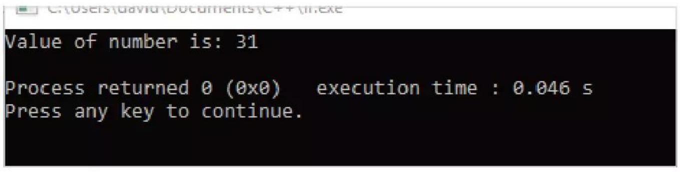
```
int num = 1; // Create integer called num with value of 1

if ( num < 30 ) // IF value of num is less than 30...
{

    cout << "The number is less than 30." << endl; // Then this output is displayed
```

STEP 4 The second cout statement displays the current value of num and the program terminates safely. It’s easy to see how the if statement works if you were to change the initial value of num from 1 to 31.



```
#include <iostream>
using namespace std;

int main  ()

{

    int num = 31; // Change the value of nu

    if ( num < 30 ) // IF value of num is l
    {
```

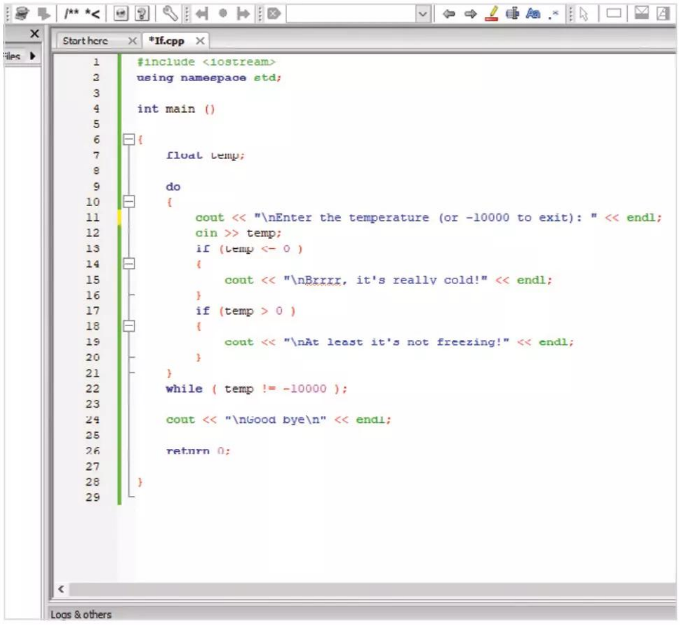
STEP 5 When you change the value to anything above 30, then build and run the code, you can see that the only line to be outputted to the screen is the second cout statement, displaying the current value of num. This is because the initial if statement is false, so it ignores the code within the braces.



```
Value of number is: 31

Process returned 0 (0x0)   execution time : 0.046 s
Press any key to continue.
```

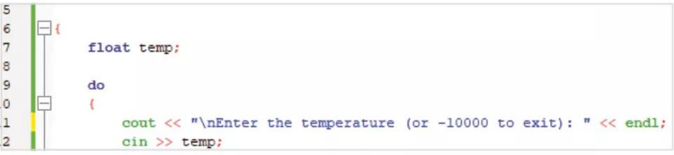
STEP 6 You can include an if statement within a do… while loop. For example:

```
{
    float temp;

    do
    {
        cout << “\nEnter the temperature (or
-10000 to exit): “ << endl;
        cin >> temp;
        if (temp <= 0 )
        {
            cout << “\nBrrrr, it’s really cold!”
<< endl;
        }
        if (temp > 0 )
        {
            cout << “\nAt least it’s not
freezing!” << endl;
        }
    }
    while ( temp != -10000 );

    cout << “\nGood bye\n” << endl;

    return 0;

}
```



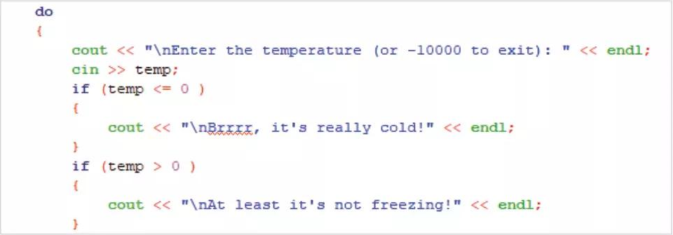
```
#include <iostream>
using namespace std;

int main ()

{
    float temp;

    do
    {
        cout << "\nEnter the temperature (or -10000 to exit): " << endl;
        cin >> temp;
        if (temp <= 0 )
        {
            cout << "\nBrrrr, it's really cold!" << endl;
        }
        if (temp > 0 )
        {
            cout << "\nAt least it's not freezing!" << endl;
        }
    }
    while ( temp != -10000 );

    cout << "\nGood bye\n" << endl;

    return 0;

}
```

STEP 7 The code in Step 6 is simplistic but effective. First we created a floating point integer called temp, then a do… while loop that asks the user to enter the current temperature.



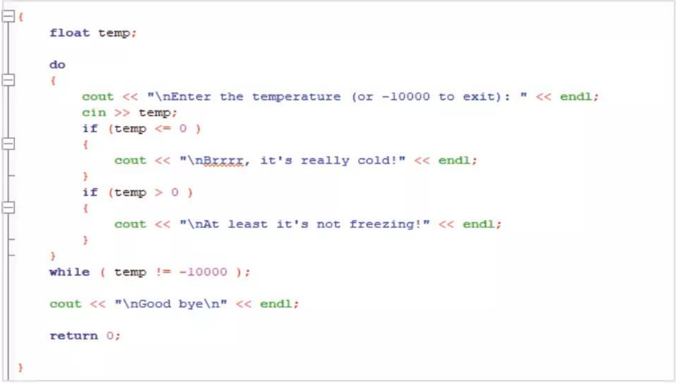
```
{
    float temp;

    do
    {
        cout << "\nEnter the temperature (or -10000 to exit): " << endl;
        cin >> temp;
```

STEP 8 The first if statement checks to see if the user’s inputted value is less than or equal to zero. If it is, then the output is ‘Brrrr, it’s really cold!’. Otherwise, if the input is greater than zero, the code outputs ‘At least it’s not freezing!’.



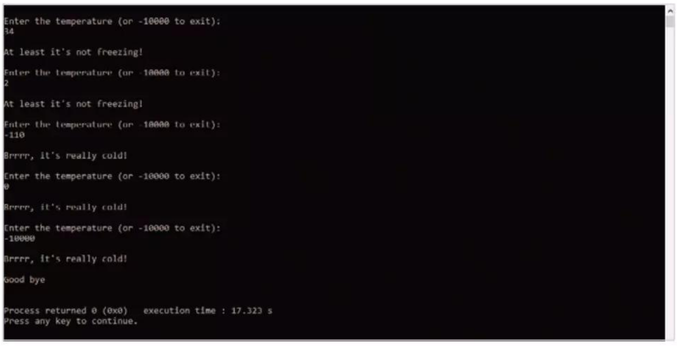
```
do
{
    cout << "\nEnter the temperature (or -10000 to exit): " << endl;
    cin >> temp;
    if (temp <= 0 )
    {
        cout << "\nBrrrr, it's really cold!" << endl;
    }
    if (temp > 0 )
    {
        cout << "\nAt least it's not freezing!" << endl;
    }
```

STEP 9 Finally, if the user enters the value -10000, which is impossibly cold so is therefore a unrealistic value, the do… while loop is terminated and a friendly ‘Good bye’ is displayed to the screen.



```
{
    float temp;

    do
    {
        cout << "\nEnter the temperature (or -10000 to exit): " << endl;
        cin >> temp;
        if (temp <= 0 )
        {
            cout << "\nBrrrr, it's really cold!" << endl;
        }
        if (temp > 0 )
        {
            cout << "\nAt least it's not freezing!" << endl;
        }
    }
    while ( temp != -10000 );

    cout << "\nGood bye\n" << endl;

    return 0;

}
```

STEP 10 Using if is quite powerful, if it’s used correctly. Just remember that if the condition is true then the code executes what’s in the braces. If not, it continues on its merry way. See what else you can come up with using if and a combination of loops.



```
Enter the temperature (or -10000 to exit):
34

At least it's not freezing!

Enter the temperature (or -10000 to exit):
2

At least it's not freezing!

Enter the temperature (or -10000 to exit):
-110

Brrrr, it's really cold!

Enter the temperature (or -10000 to exit):
0

Brrrr, it's really cold!

Enter the temperature (or -10000 to exit):
-10000

Brrrr, it's really cold!

Good bye


Process returned 0 (0x0)   execution time : 17.323 s
Press any key to continue.
```

If... Else Statement

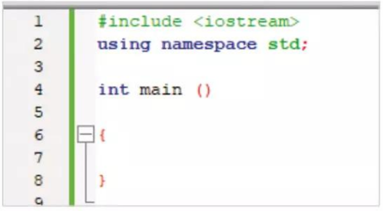
There is a much better way to use an if statement in your code, with if… else. If… else works in much the same way as a standard if statement. If the Boolean expression is true, the code with in the braces is executed. Else, the code within the next set of braces is used instead.

IF YES, ELSE NO

There are two sections of code that can be executed depending on the outcome in an if... else statement. It's quite easy to visualise once you get used to its structure.

STEP 1 Begin with a new C++ file and the standard headers:

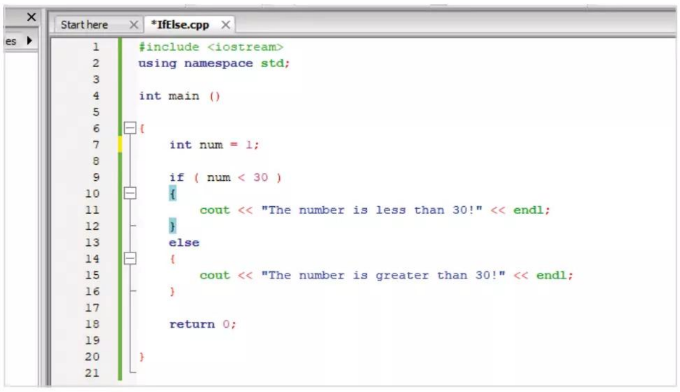
```
#include <iostream>
using namespace std;

int main ()

{

}
```



```
#include <iostream>
using namespace std;

int main ()

{

}
```

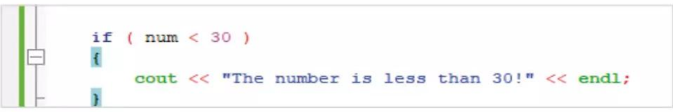
STEP 2 Let's expand the code from the If Statement on the previous page:

```
{
    int num = 1;

    if ( num < 30 )
    {
        cout << “The number is less than 30!” <<
endl;
    }
    else
    {
        cout << “The number is greater than 30!”
<< endl;
    }

    return 0;

}
```



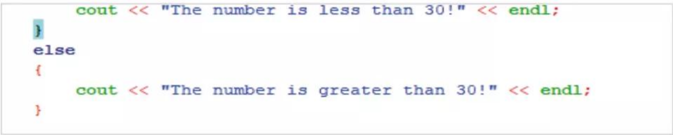
```
#include <iostream>
using namespace std;

int main ()

{
    int num = 1;

    if ( num < 30 )
    {
        cout << "The number is less than 30!" << endl;
    }
    else
    {
        cout << "The number is greater than 30!" << endl;
    }

    return 0;

}
```

STEP 3 The first line in the code creates the integer called num and gives it a value of 1. The if statement checks to see if the value of num is less than thirty and if it is it outputs "The number is less than 30!" to the console.



```
    if ( num < 30 )
    {
        cout << "The number is less than 30!" << endl;
    }
```

STEP 4 The else companion to if checks if the number is greater than 30 and if so, then displays "The number is greater than 30!" to the console; and finally, the code is terminated satisfactorily.



```
        cout << "The number is less than 30!" << endl;
    }
    else
    {
        cout << "The number is greater than 30!" << endl;
    }
```

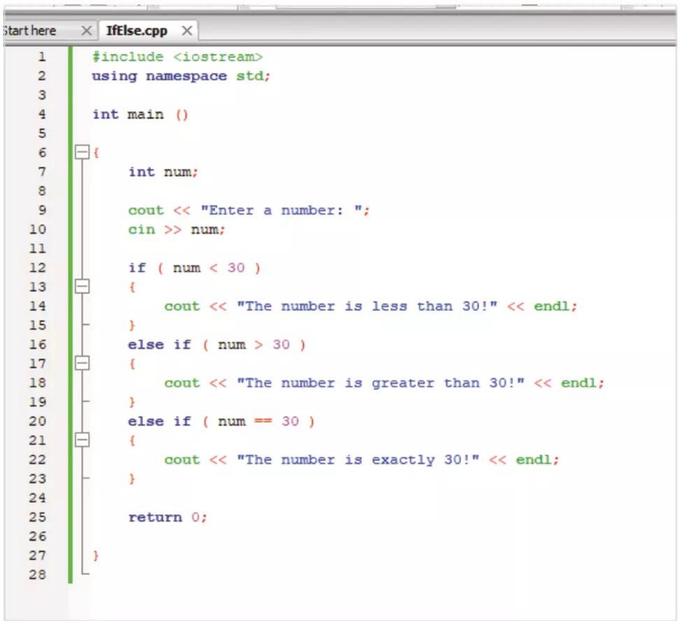
STEP 5 You can change the value of num in the code or you can improve the code by asking the user to enter a value:

```
{
    int num;

    cout << “Enter a number: “;
    cin >> num;

    if ( num < 30 )
    {
        cout << “The number is less than 30!” <<
endl;
    }
    else
    {
        cout << “The number is greater than 30!”
<< endl;
    }

    return 0;

}
```

STEP 6

The code works the same way, as you would expect, but what if you wanted to display something if the user entered the number 30? Try this:

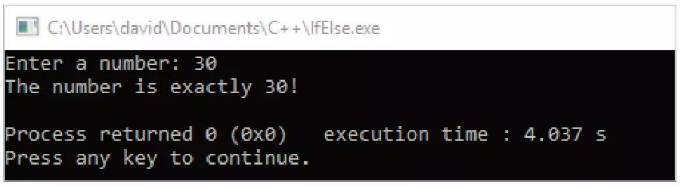
```
{
    int num;

    cout << “Enter a number: “;
    cin >> num;

    if ( num < 30 )
    {
        cout << “The number is less than 30!” <<
endl;
    }
    else if ( num > 30 )
    {
        cout << “The number is greater than 30!”
<< endl;
    }
    else if ( num == 30 )
    {
        cout << “The number is exactly 30!” <<
endl;
    }

    return 0;

}
```



```
#include <iostream>
using namespace std;

int main ()

{

    int num;

    cout << "Enter a number: ";
    cin >> num;

    if ( num < 30 )
    {
        cout << "The number is less than 30!" << endl;
    }
    else if ( num > 30 )
    {
        cout << "The number is greater than 30!" << endl;
    }
    else if ( num == 30 )
    {
        cout << "The number is exactly 30!" << endl;
    }

    return 0;

}
```

STEP 7

The new addition to the code is what's known as a nested if… else statement. This allows you to check for multiple conditions. In this case, if the user enters a number less than 30, greater than 30 or actually 30 itself, a different outcome is presented to them.



```
Enter a number: 30
The number is exactly 30!

Process returned 0 (0x0)   execution time : 4.037 s
Press any key to continue.
```

STEP 8

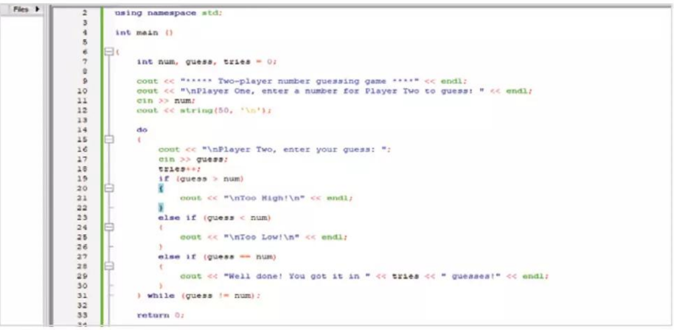
You can take this up a notch and create a two-player number guessing game. Begin by creating the variables:

```
int num, guess, tries = 0;
    cout << “***** Two-player number guessing game
****” << endl;
    cout << “\nPlayer One, enter a number for
Player Two to guess: “ << endl;
    cin >> num;
    cout << string(50, ‘\n’);
```

STEP 9

The cout << string(50, '\n') line clears the screen so Player Two doesn't see the entered number. Now you can create a do… while loop, together with if… else:

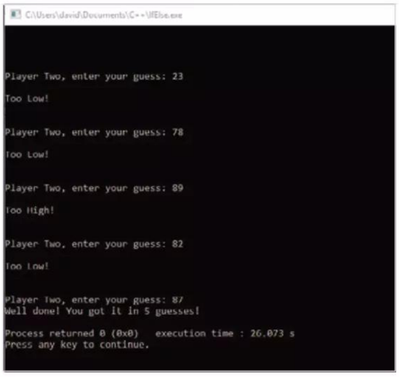
```
  do
    {
        cout << “\nPlayer Two, enter your guess: “;
        cin >> guess;
        tries++;
        if (guess > num)
        {
            cout << “\nToo High!\n” << endl;
        }
        else if (guess < num)
        {
            cout << “\nToo Low!\n” << endl;
        }
        else if (guess == num)
        {
            cout << “Well done! You got it in “ <<
tries << “ guesses!” << endl;
        }
    } while (guess != num);

    return 0;
```



```
using namespace std;

int main ()

{
    int num, guess, tries = 0;

    cout << "***** Two-player number guessing game ****" << endl;
    cout << "\nPlayer One, enter a number for Player Two to guess: " << endl;
    cin >> num;
    cout << string(50, '\n');

    do
    {
        cout << "\nPlayer Two, enter your guess: ";
        cin >> guess;
        tries++;
        if (guess > num)
        {
            cout << "\nToo High!\n" << endl;
        }
        else if (guess < num)
        {
            cout << "\nToo Low!\n" << endl;
        }
        else if (guess == num)
        {
            cout << "Well done! You got it in " << tries << " guesses!" << endl;
        }
    } while (guess != num);

    return 0;
```

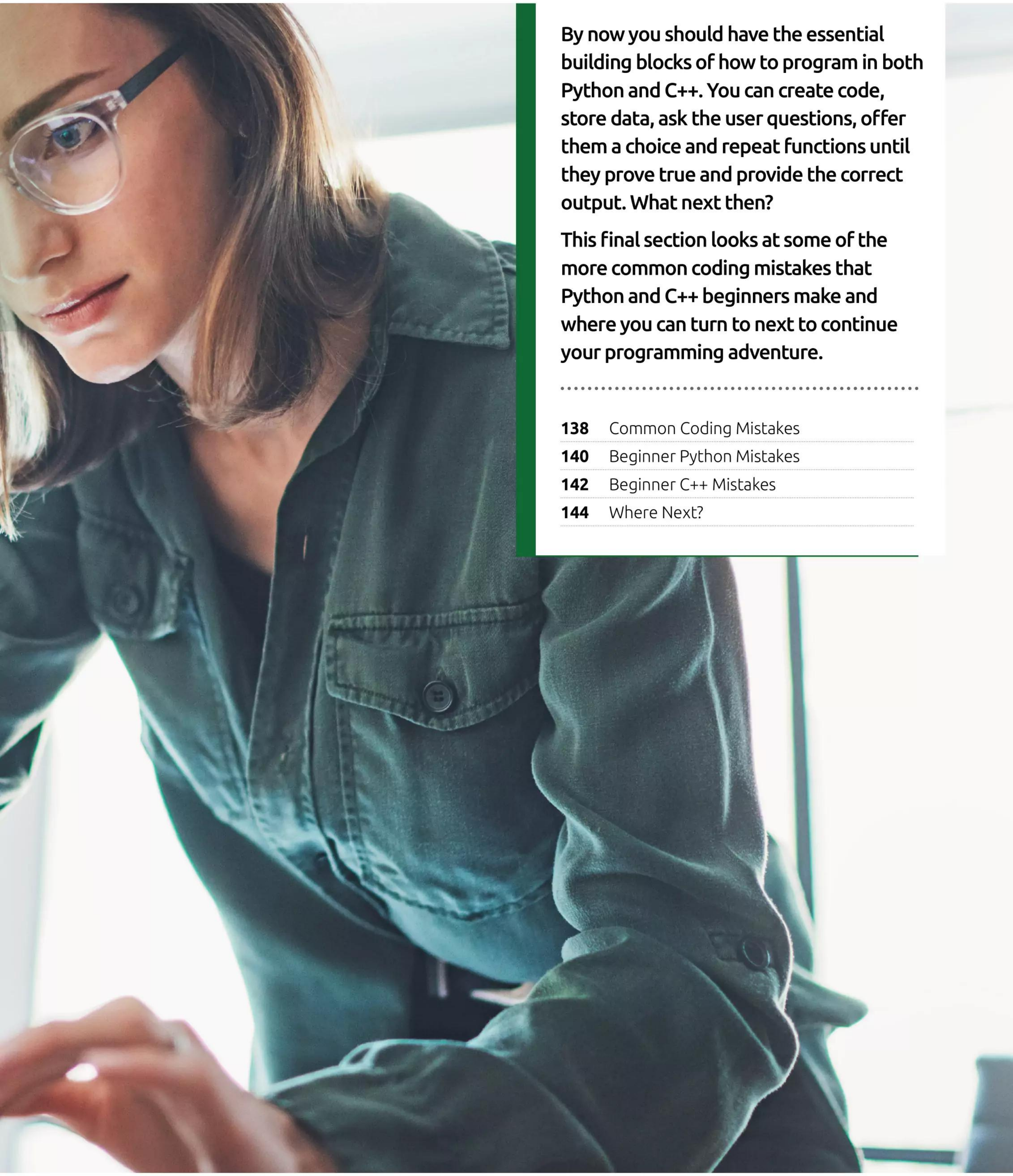
STEP 10

Grab a second player, then build and run the code. Player One enters the number to be guessed, then Player Two can take as many guesses as they need to get the right number. Want to make it harder? Maybe use decimal numbers.



```
Player Two, enter your guess: 23

Too Low!


Player Two, enter your guess: 78

Too Low!


Player Two, enter your guess: 89

Too High!


Player Two, enter your guess: 82

Too Low!


Player Two, enter your guess: 87
Well done! You got it in 5 guesses!

Process returned 0 (0x0)   execution time : 26.073 s
Press any key to continue.
```

Working with Code



By now you should have the essential building blocks of how to program in both Python and C++. You can create code, store data, ask the user questions, offer them a choice and repeat functions until they prove true and provide the correct output. What next then?

This final section looks at some of the more common coding mistakes that Python and C++ beginners make and where you can turn to next to continue your programming adventure.

138 Common Coding Mistakes
140 Beginner Python Mistakes
142 Beginner C++ Mistakes
144 Where Next?

Common Coding Mistakes

When you start something new you're inevitably going to make mistakes, this is purely down to inexperience and those mistakes are great teachers in themselves. However, even experts make the occasional mishap. Thing is, to learn from them as best you can.

X=MISTAKE, PRINT Y

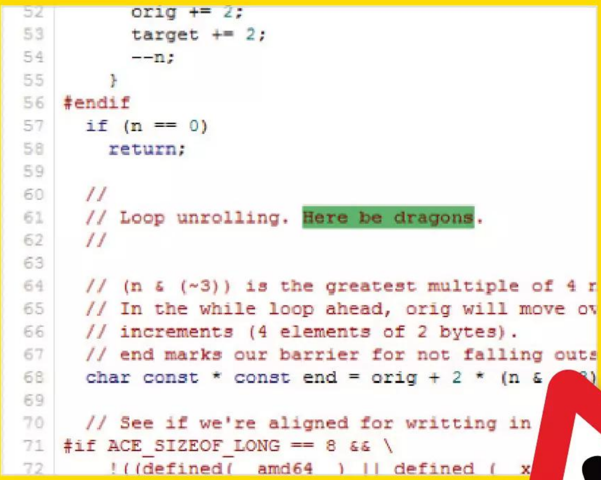
There are many pitfalls for the programmer to be aware of, far too many to be listed here. Being able to recognise a mistake and fix it is when you start to move into more advanced territory.

SMALL CHUNKS

It would be wonderful to be able to work like Neo from The Matrix movies. Simply ask, your operator loads it into your memory and you instantly know everything about the subject. Sadly though, we can't do that. The first major pitfall is someone trying to learn too much, too quickly. So take coding in small pieces and take your time.

//COMMENTS

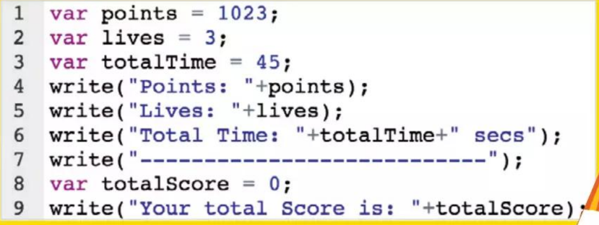
Use comments. It's a simple concept but commenting on your code saves so many problems when you next come to look over it. Inserting comment lines helps you quickly sift through the sections of code that are causing problems; also useful if you need to review an older piece of code.



```
            orig += 2;
            target += 2;
            --n;
        }
#endif
    if (n == 0)
      return;

    //
    // Loop unrolling. Here be dragons.
    //

    // (n & (~3)) is the greatest multiple of 4 r
    // In the while loop ahead, orig will move ov
    // increments (4 elements of 2 bytes).
    // end marks our barrier for not falling outs
    char const * const end = orig + 2 * (n &
    
    // See if we're aligned for writting in
#if ACE_SIZEOF_LONG == 8 && \
      !((defined(  amd64  ) || defined (  x
```

EASY VARIABLES

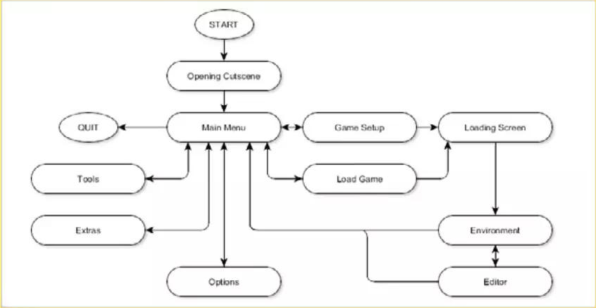
Meaningful naming for variables is a must to eliminate common coding mistakes. Having letters of the alphabet is fine but what happens when the code states there's a problem with x variable. It's not too difficult to name variables lives, money, player1 and so on.



```
var points = 1023;
var lives = 3;
var totalTime = 45;
write("Points: "+points);
write("Lives: "+lives);
write("Total Time: "+totalTime+" secs");
write("---------------------------");
var totalScore = 0;
write("Your total Score is: "+totalScore);
```

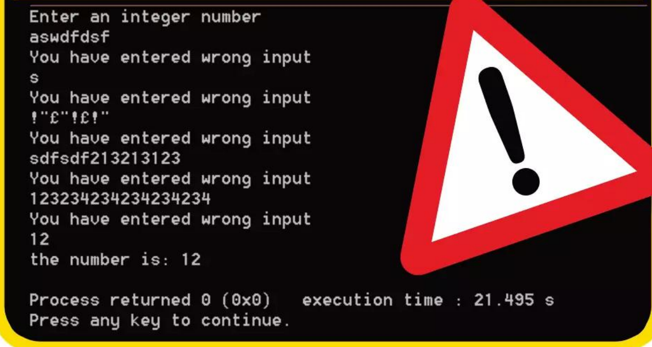
PLAN AHEAD

While it's great to wake up one morning and decide to code a classic text adventure, it's not always practical without a good plan. Small snippets of code can be written without too much thought and planning but longer and more in-depth code requires a good working plan to stick to and help iron out the bugs.



USER ERROR

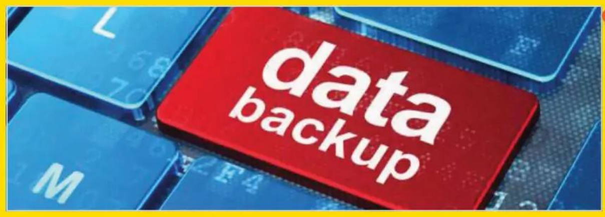
User input is often a paralysing mistake in code. For example, when the user is supposed to enter a number for their age and instead they enter it in letters. Often a user can enter so much into an input that it overflows some internal buffer, thus sending the code crashing. Watch those user inputs and clearly state what's needed from them.



```
Enter an integer number
aswdfdsf
You have entered wrong input
s
You have entered wrong input
!"£"!£!"
You have entered wrong input
sdfsdf213213123
You have entered wrong input
12323423423423423 4
You have entered wrong input
12
the number is: 12

Process returned 0 (0x0)   execution time : 21.495 s
Press any key to continue.
```

BACKUPS

Always make a backup of your work, with a secondary backup for any changes you've made. Mistakes can be rectified if there's a good backup in place to revert to for those times when something goes wrong. It's much easier to start where you left off, rather than starting from the beginning again.



RE-INVENTING WHEELS

You can easily spend days trying to fathom out a section of code to achieve a given result and it's frustrating and often time-wasting. While it's equally rewarding to solve the problem yourself, often the same code is out there on the Internet somewhere. Don't try and re-invent the wheel, look to see if some else has done it first.

```
class="multi-col-menu.html">Multiple Column Menu on Larger Viewport
<ul>
<li><a href="tall-button-header.html">Tall Button Header</a></li>
<li><a href="image-logo.html">Image Logo</a></li>
<li class="active"><a href="tall-logo.html">Tall Logo Image</a>
</ul>
</li>
<li class="has-children"> <a href="#">Carousels</a>
<ul>
<li><a href="variable-width-slider.html">Variable Image Wi
<li><a href="testimonial-slider.html">Testimonial Slider<
<li><a href="featured-work-slider.html">Featured Work Sl
<li><a href="equal-column-slider.html">Equal Column Slid
<li><a href="video-slider.html">Video Slider</a></li>
```

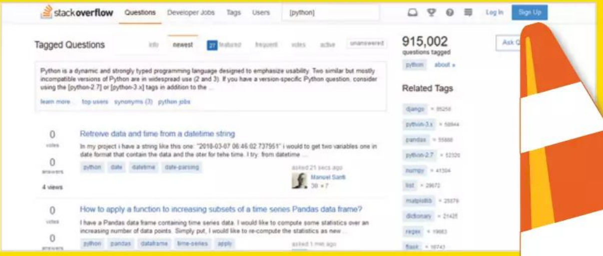
SECURE DATA

If you're writing code to deal with usernames and passwords, or other such sensitive data, then ensure that the data isn't in cleartext. Learn how to create a function to encrypt sensitive data, prior to feeding into a routine that can transmit or store it where someone may be able to get to view it.

ENCRYPT
data
security
protect
hacker
software
privilege
secure

HELP!

Asking for help is something most of us has struggled with in the past. Will the people we're asking laugh at us? Am I wasting everyone's time? It's a common mistake for someone to suffer in silence. However, as long as you ask the in the correct manner, obey any forum rules and be polite, then your question isn't silly.



915,002 questions tagged python

Python is a dynamic and strongly typed programming language designed to emphasize usability. Two similar but mostly incompatible versions of Python are in widespread use (2 and 3). If you have a version-specific Python question, consider using the [python-2.7] or [python-3.x] tags in addition to the ...

0 votes 0 answers 4 views

Retreive data and time from a datetime string

In my project i have a string like this one: "2018-03-07 06:46:02.737951" i would to get two variables one in date format that contain the data and the oter for tehe time. I try: from datetime ...

python date datetime date-parsing

asked 21 secs ago Manuel Santi 38 •7

0 votes 0 answers

How to apply a function to increasing subsets of a time series Pandas data frame?

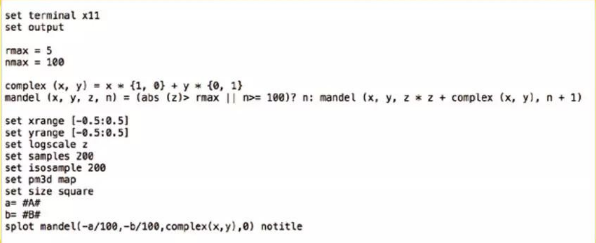
I have a Pandas data frame containing time series data. I would like to compute some statistics over an increasing number of data points. Simply put, I would like to re-compute the statistics as new ...

python pandas dataframe time-series apply

asked 1 min ago

MATHS

If your code makes multiple calculations then you need to ensure that the maths behind it is sound. There are thousands of instances where programs have offered incorrect data based on poor mathematical coding, which can have disastrous effects depending on what the code is set to do. In short, double check your code equations.



```
set terminal x11
set output

rmax = 5
nmax = 100

complex (x, y) = x * {1, 0} + y * {0, 1}
mandel (x, y, z, n) = (abs (z)> rmax || n>= 100)? n: mandel (x, y, z * z + complex (x, y), n + 1)

set xrange [-0.5:0.5]
set yrange [-0.5:0.5]
set logscale z
set samples 200
set isosample 200
set pm3d map
set size square
a= #A#
b= #B#
splot mandel(-a/100,-b/100,complex(x,y),0) notitle
```

Beginner Python Mistakes

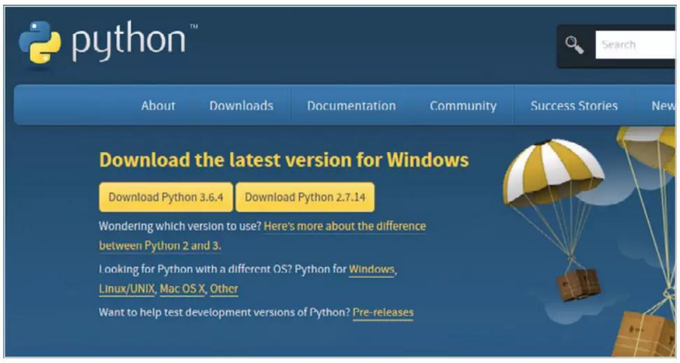
Python is a relatively easy language to get started in where there's plenty of room for the beginner to find their programming feet. However, as with any other programming language, it can be easy to make common mistakes that'll stop your code from running.

DEF BEGINNER(MISTAKES=10)

Here are ten common Python programming mistakes most beginners find themselves making. Being able to identify these mistakes will save you headaches in the future.

VERSIONS

To add to the confusion that most beginners already face when coming into programming, Python has two live versions of its language available to download and use. There is Python version 2.7.x and Python 3.6.x. The 3.6.x version is the most recent, and the one we'd recommend starting. But, version 2.7.x code doesn't always work with 3.6.x code and vice versa.



Download the latest version for Windows

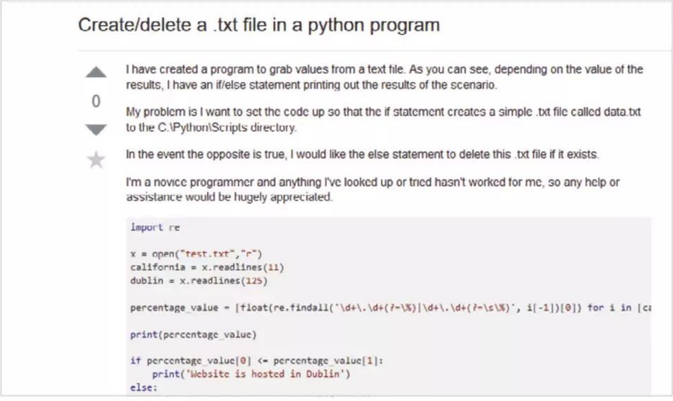
Wondering which version to use? Here's more about the difference between Python 2 and 3.

Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Pre-releases

THE INTERNET

Every programmer has and does at some point go on the Internet and copy some code to insert into their own routines. There's nothing wrong with using others' code, but you need to know how the code works and what it does before you go blindly running it on your own computer.



Create/delete a .txt file in a python program

I have created a program to grab values from a text file. As you can see, depending on the value of the results, I have an if/else statement printing out the results of the scenario.

My problem is I want to set the code up so that the if statement creates a simple .txt file called data.txt to the C:\Python\Scripts directory.

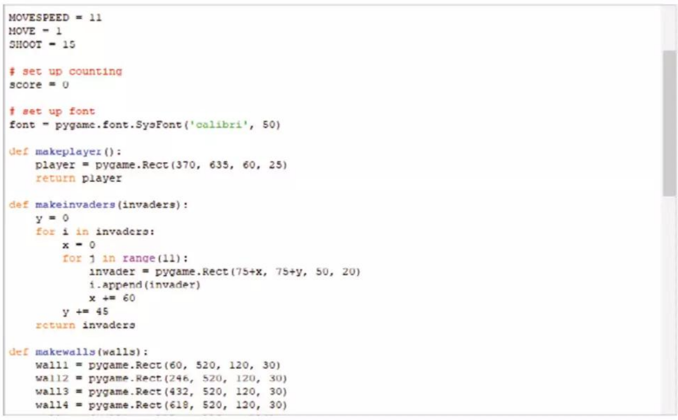
In the event the opposite is true, I would like the else statement to delete this .txt file if it exists.

I'm a novice programmer and anything I've looked up or tried hasn't worked for me, so any help or assistance would be hugely appreciated.

```
import re

x = open("test.txt","r")
california = x.readlines(11)
dublin = x.readlines(125)

percentage_value = [float(re.findall('\d+\.\d+(?=\%)|\d+\.\d+(?=\s\%)', i[-1])[0]) for i in [ca

print(percentage_value)

if percentage_value[0] <= percentage_value[1]:
    print('Website is hosted in Dublin')
else:
```

INDENTS, TABS AND SPACES

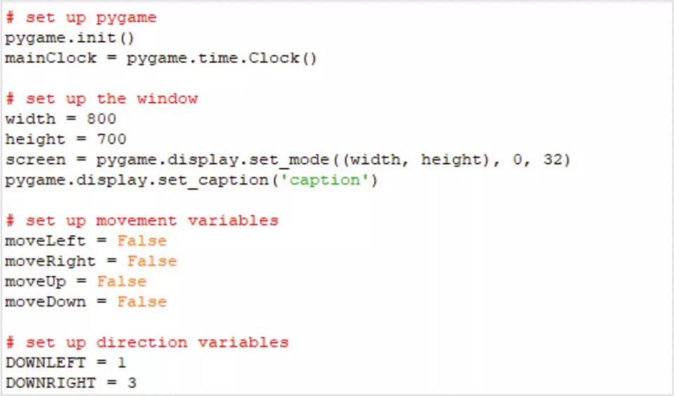
Python uses precise indentations when displaying its code. The indents mean that the code in that section is a part of the previous statement, and not something linked with another part of the code. Use four spaces to create an indent, not the Tab key.



```
MOVESPEED = 11
MOVE = 1
SHOOT = 15

# set up counting
score = 0

# set up font
font = pygame.font.SysFont('calibri', 50)

def makeplayer():
    player = pygame.Rect(370, 635, 60, 25)
    return player

def makeinvaders(invaders):
    y = 0
    for i in invaders:
        x = 0
        for j in range(11):
            invader = pygame.Rect(75+x, 75+y, 50, 20)
            i.append(invader)
            x += 60
        y += 45
    return invaders

def makewalls(walls):
    wall1 = pygame.Rect(60, 520, 120, 30)
    wall2 = pygame.Rect(246, 520, 120, 30)
    wall3 = pygame.Rect(432, 520, 120, 30)
    wall4 = pygame.Rect(618, 520, 120, 30)
```

COMMENTING

Again we mention commenting. It's a hugely important factor in programming, even if you're the only one who is ever going to view the code, you need to add comments as to what's going on. Is this function where you lose a life? Write a comment and help you, or anyone else, see what's going on.



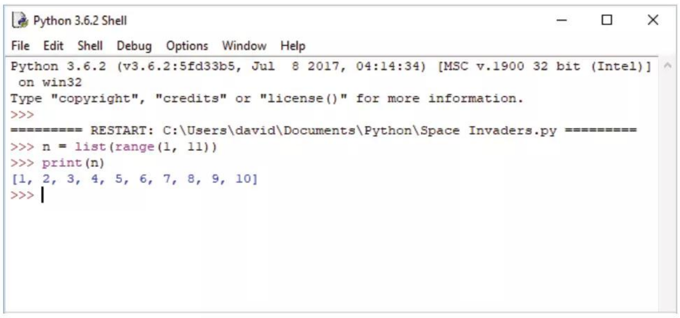
```
# set up pygame
pygame.init()
mainClock = pygame.time.Clock()

# set up the window
width = 800
height = 700
screen = pygame.display.set_mode((width, height), 0, 32)
pygame.display.set_caption('caption')

# set up movement variables
moveLeft = False
moveRight = False
moveUp = False
moveDown = False

# set up direction variables
DOWNLEFT = 1
DOWNRIGHT = 3
```

COUNTING LOOPS

Remember that in Python a loop doesn't count the last number you specify in a range. So if you wanted the loop to count from 1 to 10, then you will need to use:

```
n = list(range(1, 11))
```

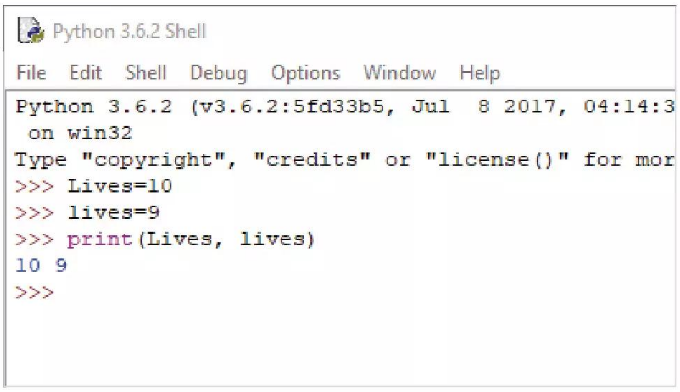
Which will return 1 to 10.



```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========= RESTART: C:\Users\david\Documents\Python\Space Invaders.py =========
>>> n = list(range(1, 11))
>>> print(n)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> 
```

CASE SENSITIVE

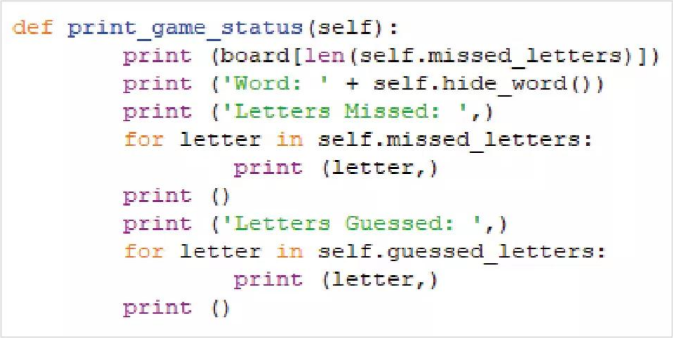
Python is a case sensitive programming language, so you will need to check any variables you assign. For example, `Lives=10` is a different variable to lives=10, calling the wrong variable in your code can have unexpected results.



```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:3
 on win32
Type "copyright", "credits" or "license()" for mor
>>> Lives=10
>>> lives=9
>>> print(Lives, lives)
10 9
>>>
```

BRACKETS

Everyone forgets to include that extra bracket they should have added to the end of the statement. Python relies on the routine having an equal amount of closed brackets to open brackets, so any errors in your code could be due to you forgetting to count your brackets; including square brackets.



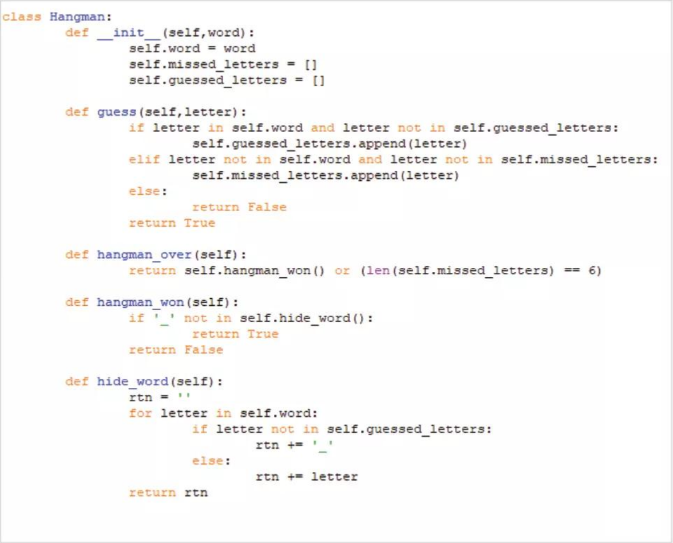
```
def print_game_status(self):
        print (board[len(self.missed_letters)])
        print ('Word: ' + self.hide_word())
        print ('Letters Missed: ',)
        for letter in self.missed_letters:
                print (letter,)
        print ()
        print ('Letters Guessed: ',)
        for letter in self.guessed_letters:
                print (letter,)
        print ()
```

COLONS

It's common for beginners to forget to add a colon to the end of a structural statement, such as:

```
class Hangman:
def guess(self, letter):
```

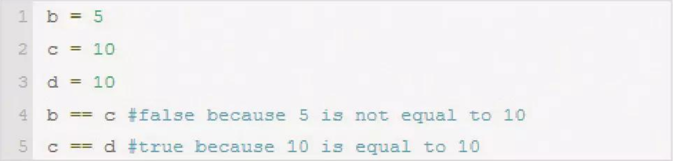
And so on. The colon is what separates the code, and creates the indents to which the following code belongs to.



```
class Hangman:
        def __init__(self,word):
                self.word = word
                self.missed_letters = []
                self.guessed_letters = []

        def guess(self,letter):
                if letter in self.word and letter not in self.guessed_letters:
                        self.guessed_letters.append(letter)
                elif letter not in self.word and letter not in self.missed_letters:
                        self.missed_letters.append(letter)
                else:
                        return False
                return True

        def hangman_over(self):
                return self.hangman_won() or (len(self.missed_letters) == 6)

        def hangman_won(self):
                if '_' not in self.hide_word():
                        return True
                return False

        def hide_word(self):
                rtn = ''
                for letter in self.word:
                        if letter not in self.guessed_letters:
                                rtn += '_'
                        else:
                                rtn += letter
                return rtn
```

OPERATORS

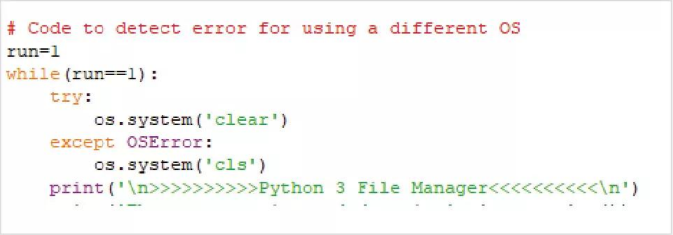
Using the wrong operator is also a common mistake to make. When you're performing a comparison between two values, for example, you need to use the equality operator (a double equals, ==). Using a single equal (=) is an assignment operator that places a value to a variable (such as, lives=10).



```
b = 5
c = 10
d = 10
b == c #false because 5 is not equal to 10
c == d #true because 10 is equal to 10
```

OPERATING SYSTEMS

Writing code for multiple platforms is difficult, especially when you start to utilise the external commands of the operating system. For example, if your code calls for the screen to be cleared, then for Windows you would use `cls`. Whereas, for Linux you need to use `clear`. You need to solve this by capturing the error and issuing it with an alternative command.



```
# Code to detect error for using a different OS
run=1
while(run==1):
    try:
        os.system('clear')
    except OSError:
        os.system('cls')
    print('\n>>>>>>>>>>Python 3 File Manager<<<<<<<<<\n')
```

Beginner C++ Mistakes

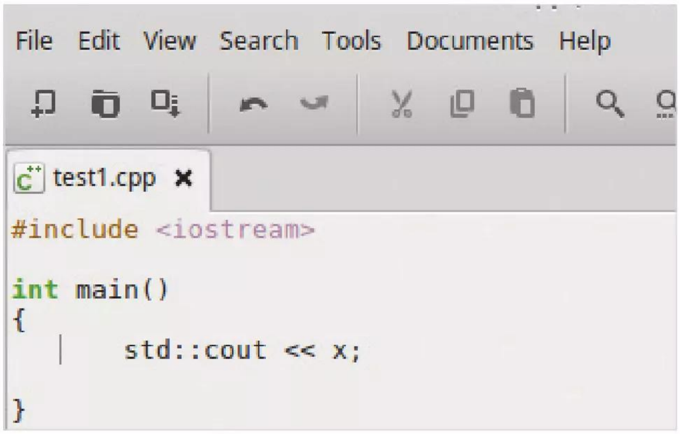
There are many pitfalls the C++ developer can encounter, especially as this is a more complex and often unforgiving language to master. Beginners need to take C++ a step at a time and digest what they've learned before moving on.

VOID(C++, MISTAKES)

Admittedly it's not just C++ beginners that make the kinds of errors we outline on these pages, even hardened coders are prone to the odd mishap here and there. Here are some common issues to try and avoid.

UNDECLARED IDENTIFIERS

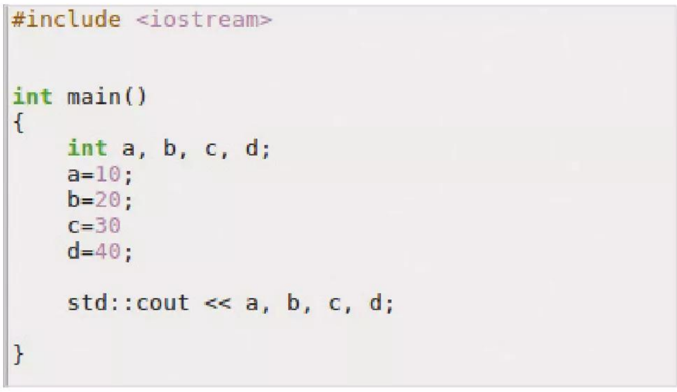
A common C++ mistake, and to be honest a common mistake with most programming languages, is when you try and output a variable that doesn't exist. Displaying the value of x on-screen is fine but not if you haven't told the compiler what the value of x is to begin with.



```
#include <iostream>

int main()
{
        std::cout << x;

}
```

SEMICOLONS

Remember that each line of a C++ program must end with a semicolon. If it doesn't then the compiler treats the line with the missing semicolon as the same line with the next semicolon on. This creates all manner of problems when trying to compile, so don't forget those semicolons.



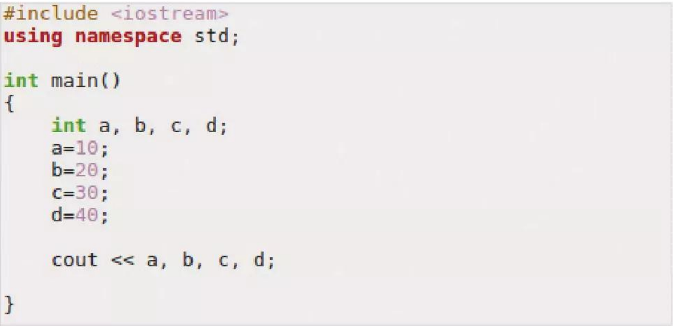
```
#include <iostream>


int main()
{
    int a, b, c, d;
    a=10;
    b=20;
    c=30
    d=40;

    std::cout << a, b, c, d;

}
```

STD NAMESPACE

Referencing the Standard Library is common for beginners throughout their code, but if you miss the std:: element of a statement, your code errors out when compiling. You can combat this by adding:

```
using namespace std;
```

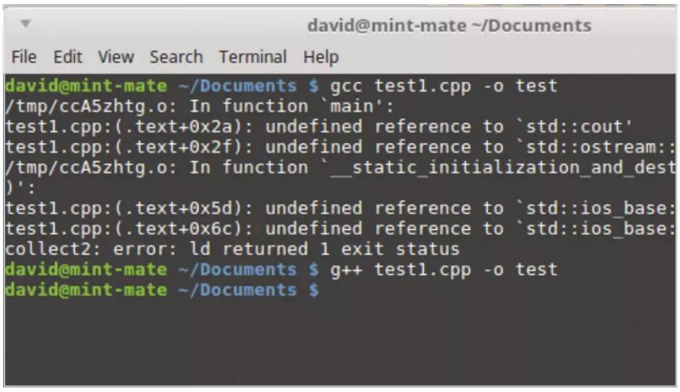
Under the #include part and simply using cout, cin and so on from then on.



```
#include <iostream>
using namespace std;

int main()
{
    int a, b, c, d;
    a=10;
    b=20;
    c=30;
    d=40;

    cout << a, b, c, d;

}
```

GCC OR G++

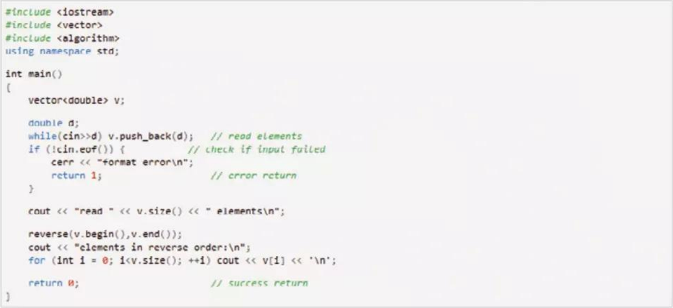
If you're compiling in Linux then you will no doubt come across gcc and g++. In short, gcc is the Gnu Compiler Collection (or Gnu C Compiler as it used to be called) and g++ is the Gnu ++ (the C++ version) of the compiler. If you're compiling C++ then you need to use g++, as the incorrect compiler drivers will be used.


david@mint-mate ~/Documents
File Edit View Search Terminal Help

```
david@mint-mate ~/Documents $ gcc test1.cpp -o test
/tmp/ccA5zhtg.o: In function `main':
test1.cpp:(.text+0x2a): undefined reference to `std::cout'
test1.cpp:(.text+0x2f): undefined reference to `std::ostream::
/tmp/ccA5zhtg.o: In function `__static_initialization_and_dest
)':
test1.cpp:(.text+0x5d): undefined reference to `std::ios_base:
test1.cpp:(.text+0x6c): undefined reference to `std::ios_base:
collect2: error: ld returned 1 exit status
david@mint-mate ~/Documents $ g++ test1.cpp -o test
david@mint-mate ~/Documents $
```

COMMENTS (AGAIN)

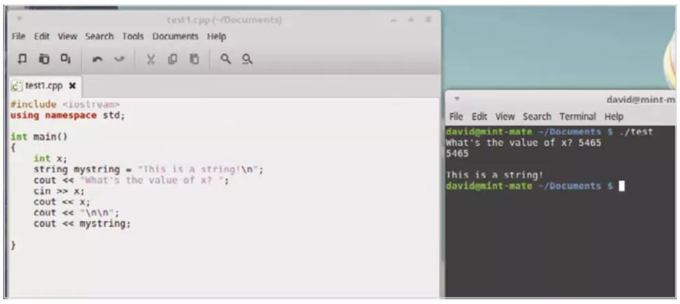
Indeed the mistake of never making any comments on code is back once more. As we've previously bemoaned, the lack of readable identifiers throughout the code makes it very difficult to look back at how it worked, for both you and someone else. Use more comments.



```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main()
{
    vector<double> v;

    double d;
    while(cin>>d) v.push_back(d);   // read elements
    if (!cin.eof()) {              // check if input failed
        cerr << "format error\n";
        return 1;                  // error return
    }

    cout << "read " << v.size() << " elements\n";

    reverse(v.begin(),v.end());
    cout << "elements in reverse order:\n";
    for (int i = 0; i<v.size(); ++i) cout << v[i] << '\n';

    return 0;                      // success return
}
```

QUOTES

Missing quotes is a common mistake to make, for every level of user. Remember that quotes need to encase strings and anything that's going to be outputted to the screen or into a file, for example. Most compilers errors are due to missing quotes in the code.



test1.cpp (~/Documents)
File Edit View Search Tools Documents Help
test1.cpp
david@mint-m
File Edit View Search Terminal Help

```
#include <iostream>
using namespace std;

int main()
{
    int x;
    string mystring = "This is a string!\n";
    cout << "What's the value of x? ";
    cin >> x;
    cout << x;
    cout << "\n\n";
    cout << mystring;

}
```

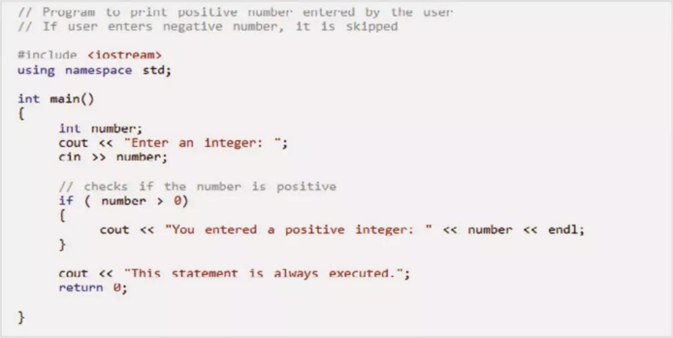
```
david@mint-mate ~/Documents $ ./test
What's the value of x? 5465
5465

This is a string!
david@mint-mate ~/Documents $
```

EXTRA SEMICOLONS

While it's necessary to have a semicolon at the end of every C++ line, there are some exceptions to the rule. Semicolons need to be at the end of every complete statement but some lines of code aren't complete statements. Such as:

```
#include
if lines
switch lines
```

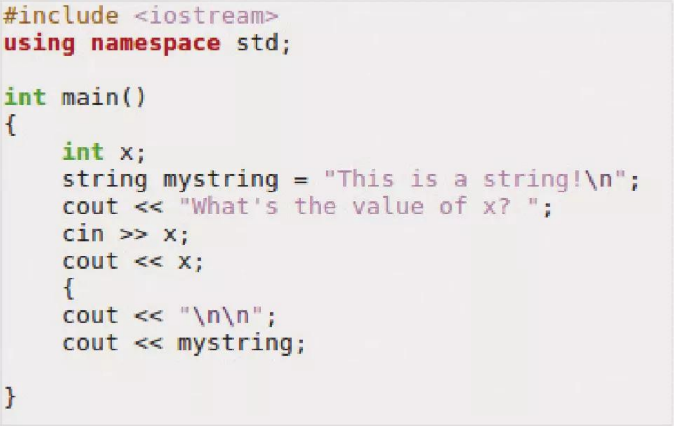
If it sounds confusing don't worry, the compiler lets you know where you went wrong.



```
// Program to print positive number entered by the user
// If user enters negative number, it is skipped

#include <iostream>
using namespace std;

int main()
{
     int number;
     cout << "Enter an integer: ";
     cin >> number;

     // checks if the number is positive
     if ( number > 0)
     {
          cout << "You entered a positive integer: " << number << endl;
     }

     cout << "This statement is always executed.";
     return 0;

}
```

TOO MANY BRACES

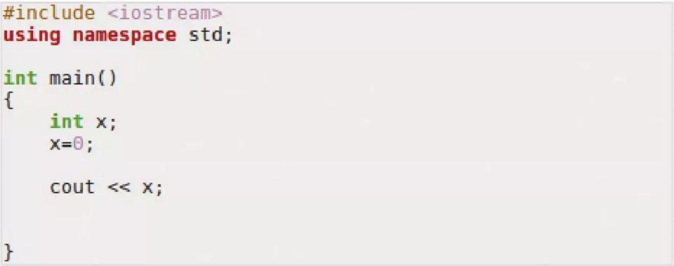
The braces, or curly brackets, are beginning and ending markers around blocks of code. So for every { you must have a }. Often it's easy to include or miss out one or the other facing brace when writing code; usually when writing in a text editor, as an IDE adds them for you.



```
#include <iostream>
using namespace std;

int main()
{
    int x;
    string mystring = "This is a string!\n";
    cout << "What's the value of x? ";
    cin >> x;
    cout << x;
    {
    cout << "\n\n";
    cout << mystring;

}
```

INITIALISE VARIABLES

In C++ variables aren't initialised to zero by default. This means if you create a variable called x then, potentially, it is given a random number from 0 to 18,446,744,073,709,551,616, which can be difficult to include in an equation. When creating a variable, give it the value of zero to begin with: `x=0`.



```
#include <iostream>
using namespace std;

int main()
{
    int x;
    x=0;

    cout << x;

}
```

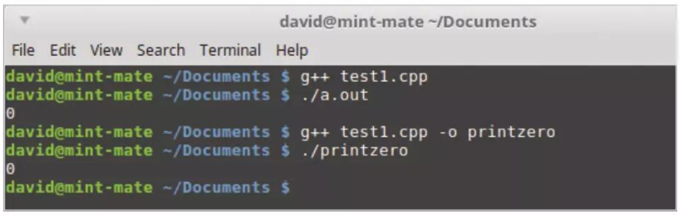
A.OUT

A common mistake when compiling in Linux is forgetting to name your C++ code post compiling. When you compile from the Terminal, you enter:

```
g++ code.cpp
```

This compiles the code in the file code.cpp and create an a.out file that can be executed with ./a.out. However, if you already have code in a.out then it's overwritten. Use:

```
g++ code.cpp -o nameofprogram
```



david@mint-mate ~/Documents
File Edit View Search Terminal Help

```
david@mint-mate ~/Documents $ g++ test1.cpp
david@mint-mate ~/Documents $ ./a.out
0
david@mint-mate ~/Documents $ g++ test1.cpp -o printzero
david@mint-mate ~/Documents $ ./printzero
0
david@mint-mate ~/Documents $
```

Where Next?

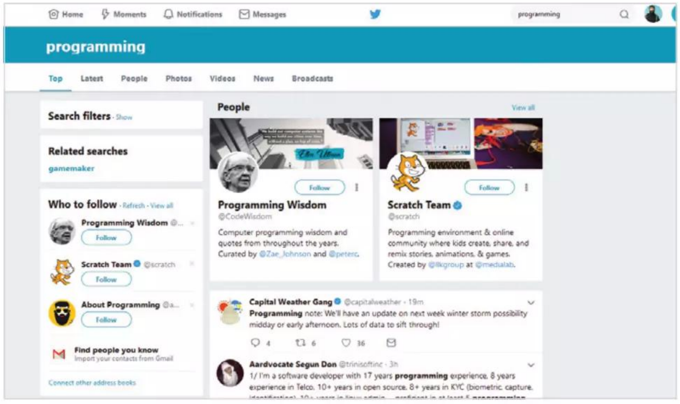
Coding, like most subjects, is a continual learning experience. You may not class yourself as a beginner any more but you still need to test your code, learn new tricks and hacks to make it more efficient and even branch out and learn another programming language.

#INCLUDE<KEEP ON LEARNING>

What can you do to further your skills, learn new coding practises, experiment and present your code and even begin to help others using what you've experienced so far?

TWITTER

Twitter isn't all trolls and antagonists, among the well-publicised vitriol are some genuine people who are more than willing to spread their coding knowledge. We recommend you find a few who you can relate to and follow them. Often they post great tips, hacks and fixes for common coding problems.


Home
Moments
Notifications
Messages
programming
programming
Top
Latest
People
Photos
Videos
News
Broadcasts
Search filters · Show
Related searches
gamemaker
Who to follow · Refresh · View all
Programming Wisdom
Follow
Scratch Team @scratch
Follow
About Programming
Follow
Find people you know
Import your contacts from Gmail
Connect other address books
People
View all
Follow
Follow

Programming Wisdom
@CodeWisdom
Computer programming wisdom and quotes from throughout the years. Curated by @Zae_Johnson and @peterc.

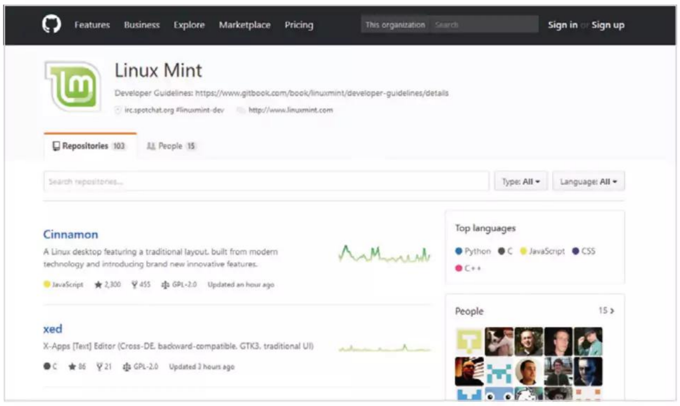
Scratch Team
@scratch
Programming environment & online community where kids create, share, and remix stories, animations, & games. Created by @llkgroup at @medialab.

Capital Weather Gang @capitalweather · 19m
Programming note: We'll have an update on next week winter storm possibility midday or early afternoon. Lots of data to sift through!
4 6 36

Aardvocate Segun Don @trinisoftinc · 3h
1/ I'm a software developer with 17 years **programming** experience. 8 years experience in Telco. 10+ years in open source. 8+ years in KYC (biometric capture,

OPEN PROJECTS

Look for open source projects that you like the sound of and offer to contribute to the code to keep it alive and up to date. There are millions of projects to choose from, so contact a few and see where they need help. It may only be a minor code update but it's a noble occupation for coders to get into.


Features
Business
Explore
Marketplace
Pricing
This organization
Search
Sign in or Sign up
Repositories 103
People 15
Search repositories...
Type: All
Language: All
Top languages
Python
C
JavaScript
CSS
C++
People
15

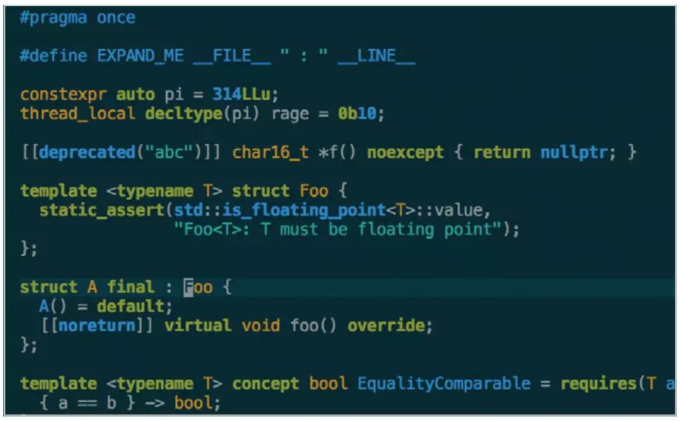
Linux Mint
Developer Guidelines: https://www.gitbook.com/book/linuxmint/developer-guidelines/details
irc.spotchat.org #linuxmint-dev
http://www.linuxmint.com

Cinnamon
A Linux desktop featuring a traditional layout, built from modern technology and introducing brand new innovative features.
JavaScript ★ 2,300 455 GPL-2.0 Updated an hour ago

xed
X-Apps [Text] Editor (Cross-DE, backward-compatible, GTK3, traditional UI)
C ★ 86 21 GPL-2.0 Updated 2 hours ago

KEEP CODING

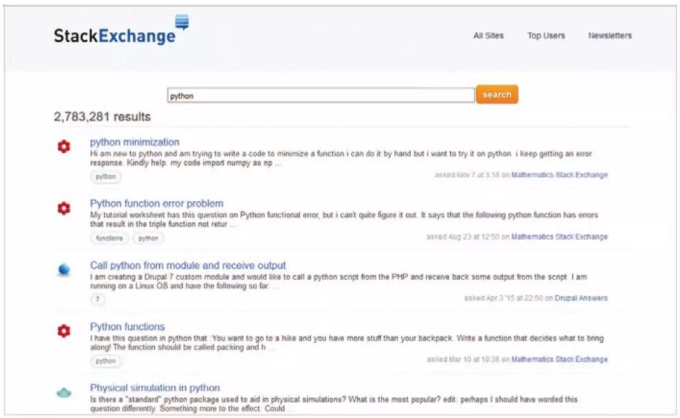
If you've mastered Python fairly well, then turn your attention to C++ or even C#. Still keep your Python skills going but learning a new coding language keeps the old brain ticking over nicely and give you a view into another community, and how they do things differently.



```
#pragma once

#define EXPAND_ME __FILE__ " : " __LINE__

constexpr auto pi = 314LLu;
thread_local decltype(pi) rage = 0b10;

[[deprecated("abc")]] char16_t *f() noexcept { return nullptr; }

template <typename T> struct Foo {
  static_assert(std::is_floating_point<T>::value,
                "Foo<T>: T must be floating point");
};

struct A final : Foo {
  A() = default;
  [[noreturn]] virtual void foo() override;
};

template <typename T> concept bool EqualityComparable = requires(T a
  { a == b } -> bool;
```

SHARE SKILLS

Become more active on coding and development knowledge sites, such as StackExchange. If you have the skills to start and help others out, not only will you feel really good for doing so but you can also learn a lot yourself by interacting with other members.


StackExchange
All Sites
Top Users
Newsletters
python
search

2,783,281 results

python minimization
Hi am new to python and am trying to write a code to minimize a function i can do it by hand but i want to try it on python. i keep getting an error response. Kindly help. my code import numpy as np ...
python
asked Nov 7 at 3:18 on Mathematics Stack Exchange

Python function error problem
My tutorial worksheet has this question on Python functional error, but i can't quite figure it out. It says that the following python function has errors that result in the triple function not retur ...
functions python
asked Aug 23 at 12:50 on Mathematics Stack Exchange

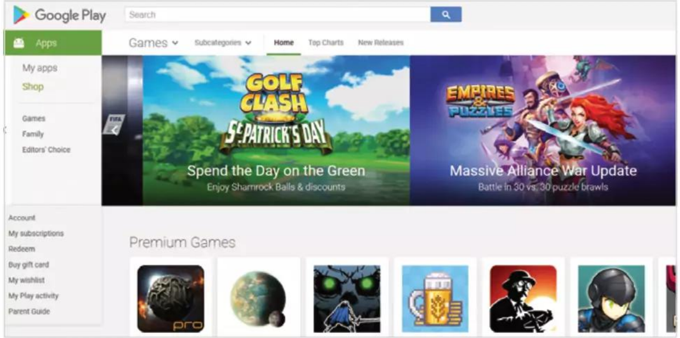
Call python from module and receive output
I am creating a Drupal 7 custom module and would like to call a python script from the PHP and receive back some output from the script. I am running on a Linux OS and have the following so far: ...
7
asked Apr 3 '15 at 22:50 on Drupal Answers

Python functions
I have this question in python that : You want to go to a hike and you have more stuff than your backpack. Write a function that decides what to bring along! The function should be called packing and h ...
python
asked Mar 10 at 18:36 on Mathematics Stack Exchange

Physical simulation in python
Is there a "standard" python package used to aid in physical simulations? What is the most popular? edit: perhaps I should have worded this question differently. Something more to the effect: Could ...

GOING MOBILE

The mobile market is a great place to test your coding skills and present any games or apps you've created. If your app is good, then who knows, it could be the next great thing to appear on the app stores. It's a good learning experience nevertheless, and something worth considering.



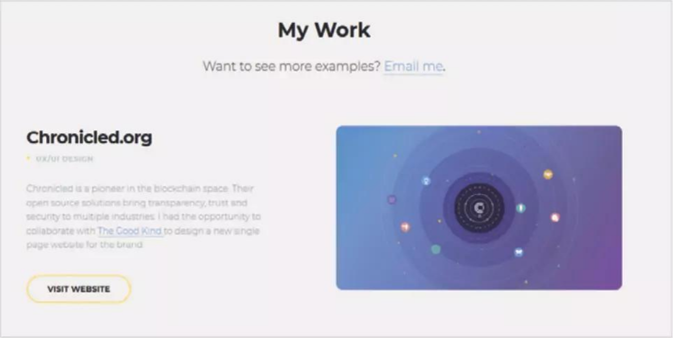
Google Play
Search
Apps
My apps
Shop
Games
Family
Editors' Choice
Account
My subscriptions
Redeem
Buy gift card
My wishlist
My Play activity
Parent Guide
Games
Subcategories
Home
Top Charts
New Releases
GOLF CLASH
ST. PATRICK'S DAY
EMPIRES & PUZZLES
Premium Games

Spend the Day on the Green
Enjoy Shamrock Balls & discounts

Massive Alliance War Update
Battle in 30 vs. 30 puzzle brawls

PORTFOLIOS

If you've learned how to code with an eye for a developer job in the future, then it's worth starting to build up an online portfolio of code. Look at job postings and see what skills they require, then learn and code something with those skills and add it to the portfolio. When it comes to applying, include a link to the portfolio.



My Work

Want to see more examples? Email me.

Chronicled.org

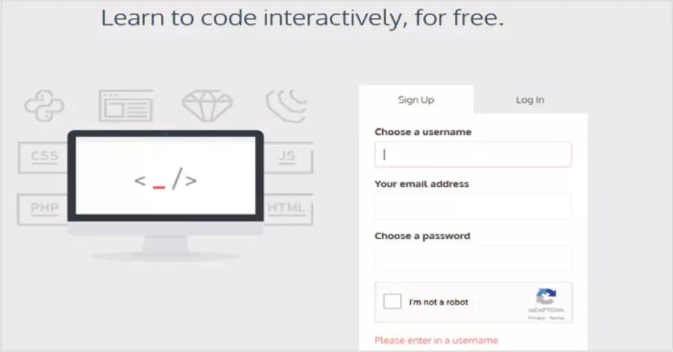
UX/UI DESIGN

Chronicled is a pioneer in the blockchain space. Their open source solutions bring transparency, trust and security to multiple industries. I had the opportunity to collaborate with The Good Kind to design a new single page website for the brand.

VISIT WEBSITE

ONLINE LEARNING

Online courses are good examples of where to take your coding skills next, even if you start from the beginner level again. Often, an online course follows a strict coding convention, so if you're self-taught then it might be worth seeing how other developers lay out their code, and what's considered acceptable.



Learn to code interactively, for free.

CSS
JS
PHP
HTML
<_/>
Sign Up
Log In

Choose a username

Your email address

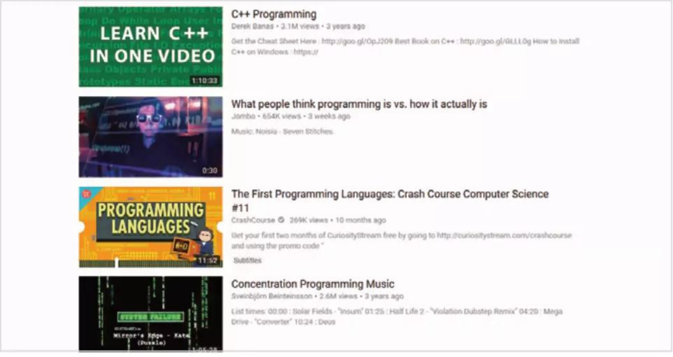
Choose a password

☐ I'm not a robot

Please enter in a username

TEACH CODE

Can you teach? If your coding skills are spot on, consider approaching a college or university to see if they have need for a programming language teacher, perhaps a part-time or evening course. If not teaching, then consider creating your own YouTube how to code channel.



LEARN C++
IN ONE VIDEO
1:10:33
0:30
PROGRAMMING LANGUAGES
11:52

C++ Programming
Derek Banas • 3.1M views • 3 years ago
Get the Cheat Sheet Here : http://goo.gl/OpJ209 Best Book on C++ : http://goo.gl/GLLL0g How to Install C++ on Windows : https://

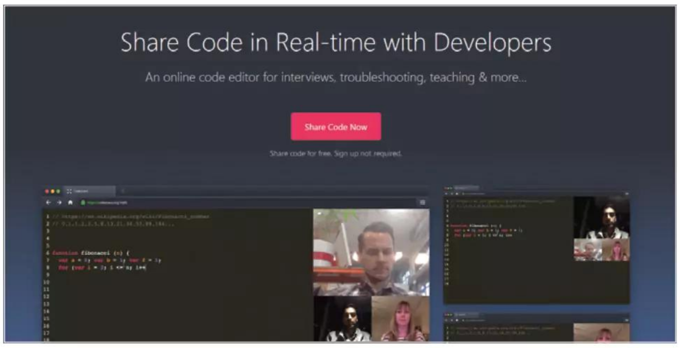
What people think programming is vs. how it actually is
Jombo • 654K views • 3 weeks ago
Music: Noisia - Seven Stitches

The First Programming Languages: Crash Course Computer Science #11
CrashCourse • 269K views • 10 months ago
Get your first two months of CuriosityStream free by going to http://curiositystream.com/crashcourse and using the promo code "
Subtitles

Concentration Programming Music
Sveinbjörn Beinteinsson • 2.6M views • 3 years ago
List times: 00:00 : Solar Fields - "Insum" 01:25 : Half Life 2 - "Violation Dubstep Remix" 04:20 : Mega Drive - "Converter" 10:24 : Deus

SHARE CODE

Get sharing, even if you think your code isn't very good. The criticism, advice and comments you receive back help you iron out any issues with your code, and you add them all to your checklist. Alternatively your code might be utterly amazing but you won't know unless you share it.



Share Code in Real-time with Developers

An online code editor for interviews, troubleshooting, teaching & more…

Share Code Now

Share code for free. Sign up not required.

HARDWARE PROJECTS

Contributing to hardware projects is a great resource for proving your code with others and learning from other contributors. Many of the developer boards have postings for coders to apply to for hardware projects, using unique code to get the most from the hardware that's being designed.

Sat, Oct 28
04:12 20 pm
13°F
Windchill: 13°F
Windspeed: calm mph
Direction: CALM
Barometer: 1012.87 Hg
Humidity: 69%
Today:
Sunday:
Monday:
Tuesday:
20°F / 17°F 0%
19°F / 10°F 100%
WINDY 13°F / 8°F 20%
14°F / 6°F 0%

Black Dog Media

Master Your Tech

From Beginner to Expert

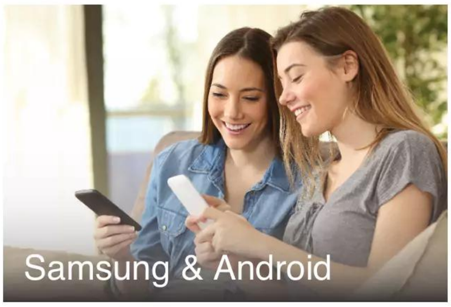
To continue learning more about your tech visit us at:

www.bdmpublications.com

FREE Tech Guides

Apple iPhone, iPad, Mac, MacBook & Watch

PC & Windows 10


Samsung & Android

Photography, Photoshop & Lightroom

Coding Python, Raspberry Pi & Linux

EXCLUSIVE Offers on our Tech Guidebooks

- Print & digital editions
- Featuring the very latest updates
- Step-by-step tutorials and guides
- Created by BDM experts

Check out our latest titles today!

The iOS 14 iPhone GuideBook
Raspberry Pi Tips, Tricks & Fixes
Coding for Beginners
The python Coding Manual
The iPad GuideBook 14
Windows 10 The Complete Guide
Photography The Beginners' Guide
The Adobe Photoshop Manual

BDM's Ultimate Photoshop

bdmpublications.com/ultimate-photoshop

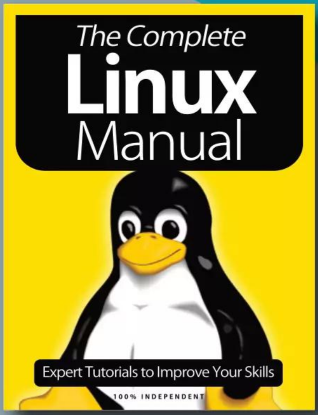
Buy our Photoshop guides and download tutorial images for free! *Simply sign up and get creative.*

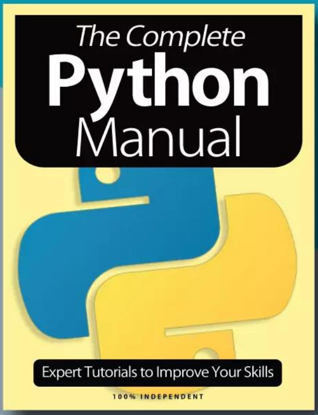
PLUS

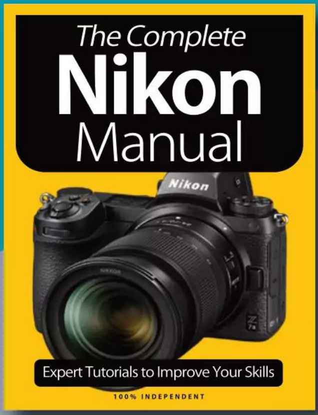
SPECIAL DEALS and Bonus Content

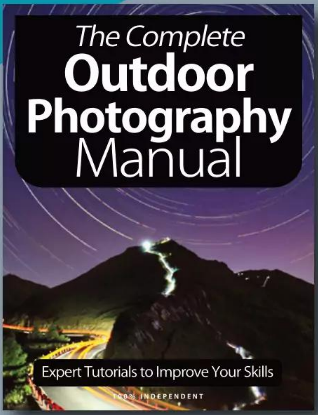
Sign up to our monthly newsletter and get the latest updates, offers and news from BDM. We are here to help you Master Your Tech!

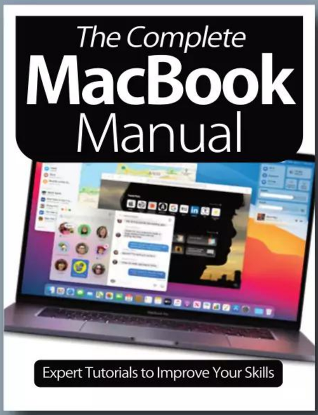
Discover more of our guides today...

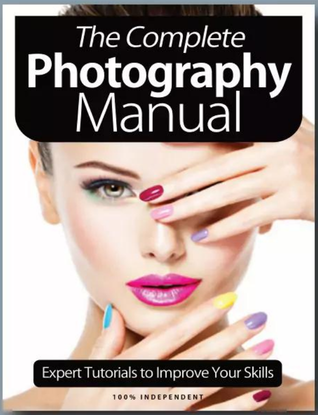

The Complete
Linux
Manual
Expert Tutorials to Improve Your Skills
100% INDEPENDENT

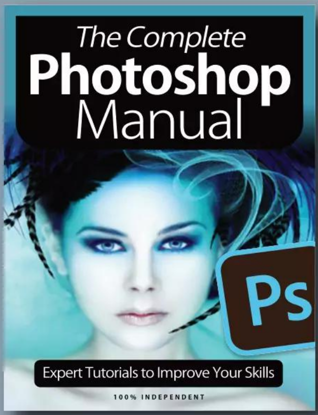

The Complete
Python
Manual
Expert Tutorials to Improve Your Skills
100% INDEPENDENT

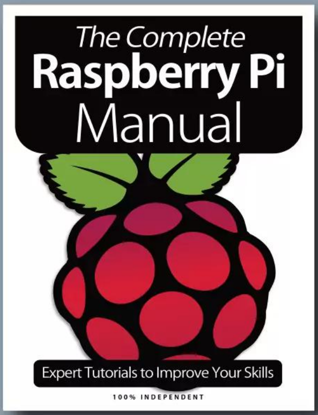

The Complete
Nikon
Manual
Expert Tutorials to Improve Your Skills
100% INDEPENDENT

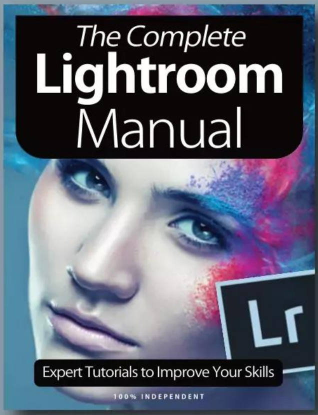

The Complete
Outdoor Photography
Manual
Expert Tutorials to Improve Your Skills
100% INDEPENDENT

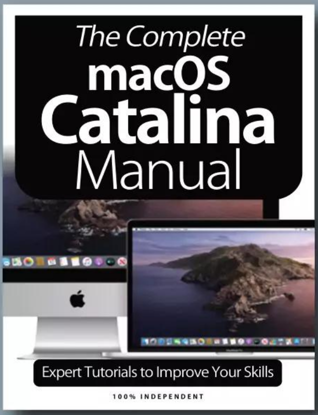

The Complete
MacBook
Manual
Expert Tutorials to Improve Your Skills

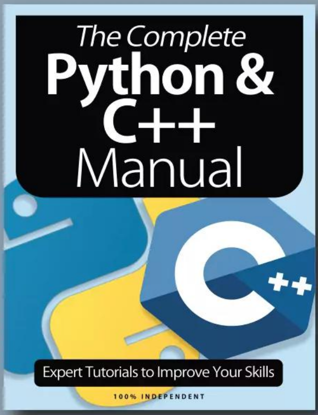

The Complete
Photography
Manual
Expert Tutorials to Improve Your Skills
100% INDEPENDENT

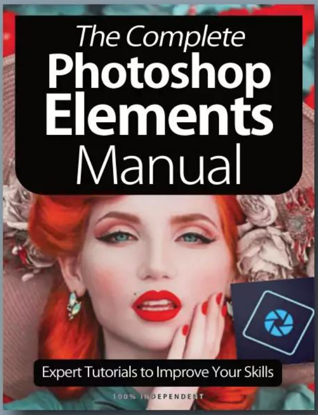

The Complete
Photoshop
Manual
Ps
Expert Tutorials to Improve Your Skills
100% INDEPENDENT


The Complete
Raspberry Pi
Manual
Expert Tutorials to Improve Your Skills
100% INDEPENDENT

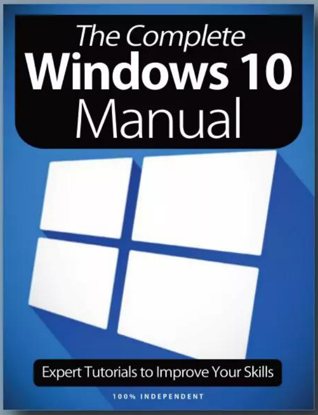

The Complete
Lightroom
Manual
Lr
Expert Tutorials to Improve Your Skills
100% INDEPENDENT

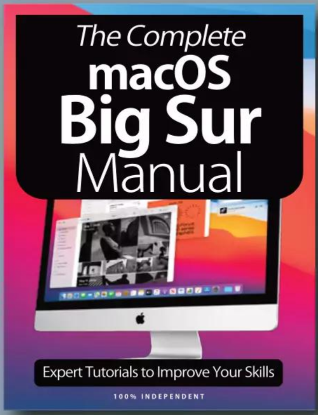

The Complete
macOS Catalina
Manual
Expert Tutorials to Improve Your Skills
100% INDEPENDENT

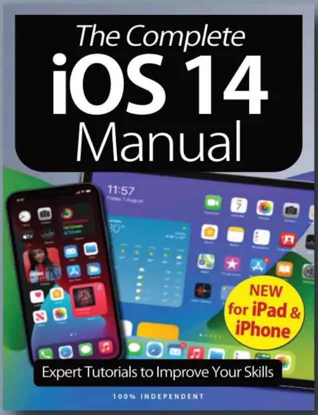

The Complete
Python & C++
Manual
Expert Tutorials to Improve Your Skills
100% INDEPENDENT


The Complete
Photoshop Elements
Manual
Expert Tutorials to Improve Your Skills
100% INDEPENDENT

The Complete
Samsung Galaxy
Manual
Expert Tutorials to Improve Your Skills
100% INDEPENDENT


The Complete
Windows 10
Manual
Expert Tutorials to Improve Your Skills
100% INDEPENDENT


The Complete
macOS Big Sur
Manual
Expert Tutorials to Improve Your Skills
100% INDEPENDENT


The Complete
iOS 14
Manual
NEW for iPad & iPhone
Expert Tutorials to Improve Your Skills
100% INDEPENDENT